



AI 应用（AI Agent）开发新范式

计缘

阿里云智能云原生应用平台

01 AI 应用概述

从“工具”到“智能伙伴”的进化



被动的命令处理工具



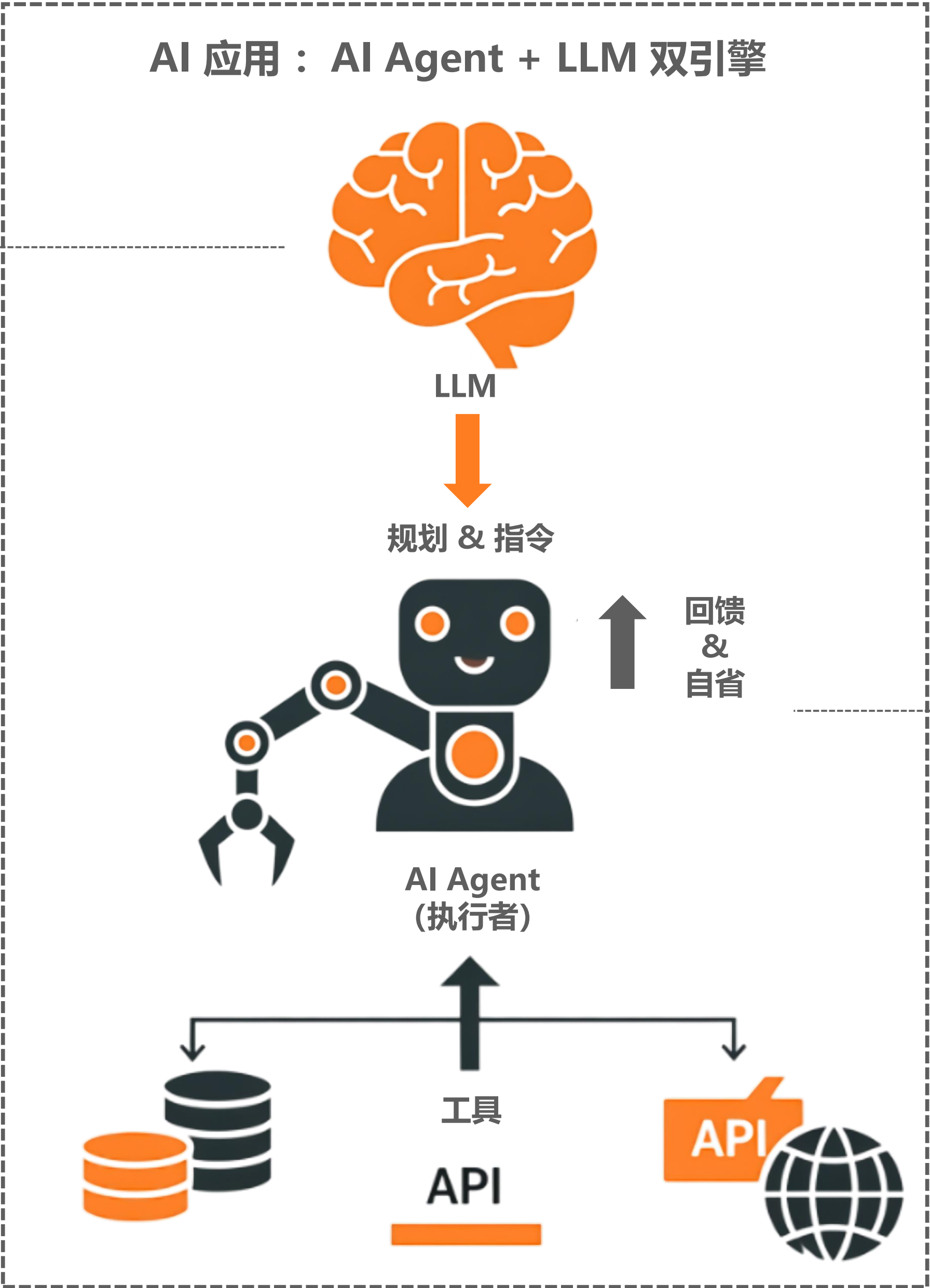
智能伙伴



AI Agent + LLM的双引擎模式

LLM扮演着认知核心，也就是“大脑”的角色。它负责处理所有与“思考”相关的任务：

- **理解意图：** 当用户用自然语言提出复杂需求时，LLM负责精准地理解其背后的真实意图。
- **规划任务：** 它能将一个模糊的目标（如“分析销售数据”）分解成一系列清晰、有序的步骤。



AI Agent赋予了LLM“手和脚”，让“思考”得以转化为“行动”。如果说LLM负责“思考做什么”，那么AI Agent则负责“如何去完成”：

- **工具调用：** 这是AI Agent最关键的能力。它可以根据LLM的规划，去调用各种外部工具来执行任务，例如查询数据库、调用公司内部系统的API、访问互联网、读写文件等。
- **任务执行与编排：** Agent负责管理整个任务流程，确保LLM规划的步骤被逐一、准确地执行。
- **与环境交互：** 它能将执行结果（如数据库查询返回的数据）反馈给LLM，供其进行下一步的思考和决策，形成一个“思考-行动-观察-再思考”的闭环。

企业能力的核心 - MCP服务



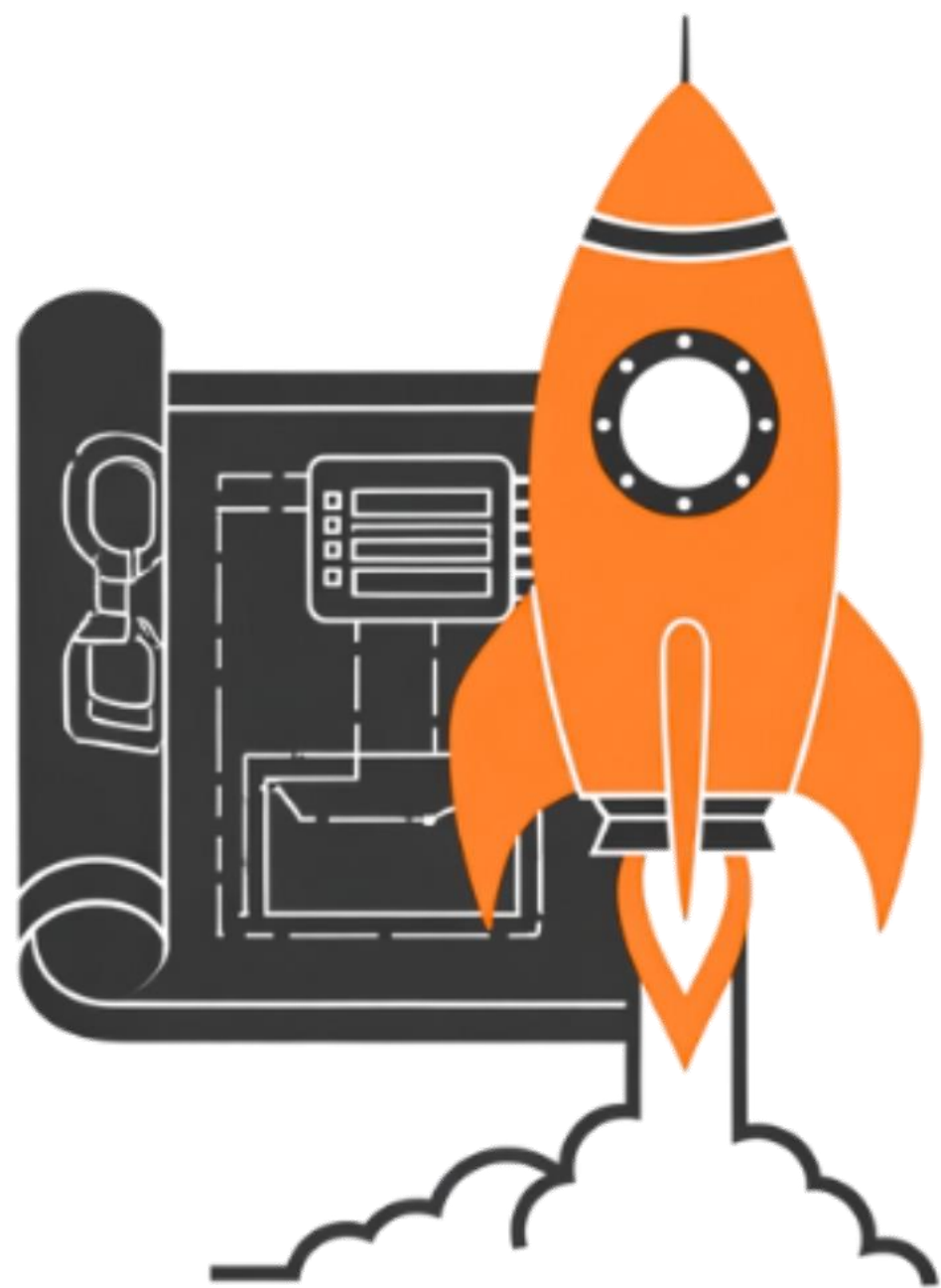
MCP的出现，很好的解决了构建AI Agent技能系统的痛点问题：

- 规范化了多者的协同关系：MCP协议规范约束了用户、AI Agent、LLM、后端服务四者之间的系统关系。
- AI Agent和后端服务快速对接：无需后端服务改造，也无需AI Agent改造，无需了解和解析后端服务接口的返回格式。

MCP服务是企业AI应用的基石。它将企业零散的IT资产和服务，转化为AI可以理解和调用的标准化能力，从而为上层的AI Agent源源不断地输送技能。

构建AI应用的两种路径：全新开发 vs. 存量改造

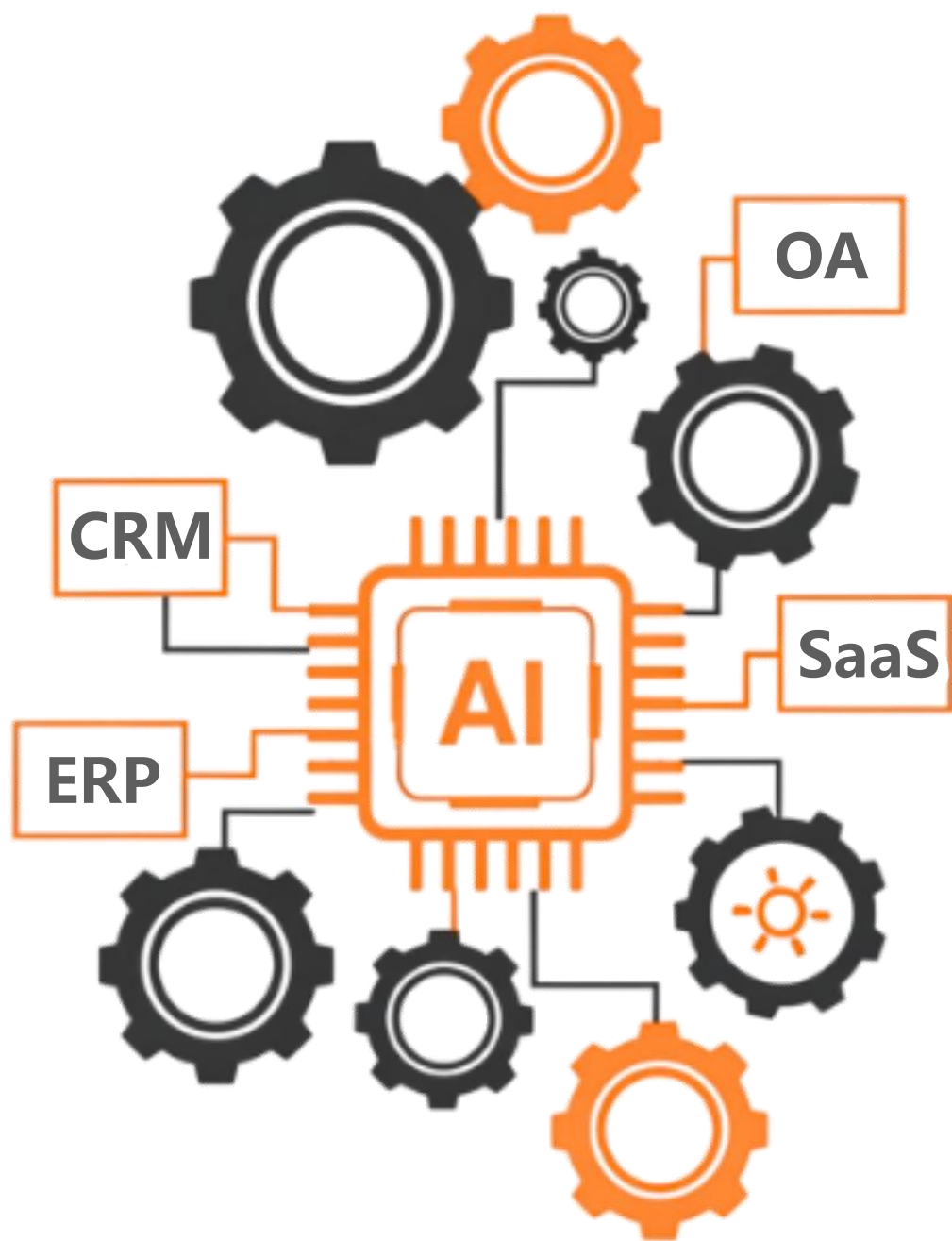
Brand New Development



全新开发：开创业务新大陆

这指的是从零开始，为一个全新的业务场景或颠覆性的产品构想，原生设计和开发AI应用。这种模式不受历史技术债务的束缚，可以采用最先进的架构，最大化地发挥AI Agent的能力，是实现颠覆式创新的最佳路径。例如，打造一个面向金融行业的AI研究分析师，或者开发一个企业内部的“超级知识入口”。

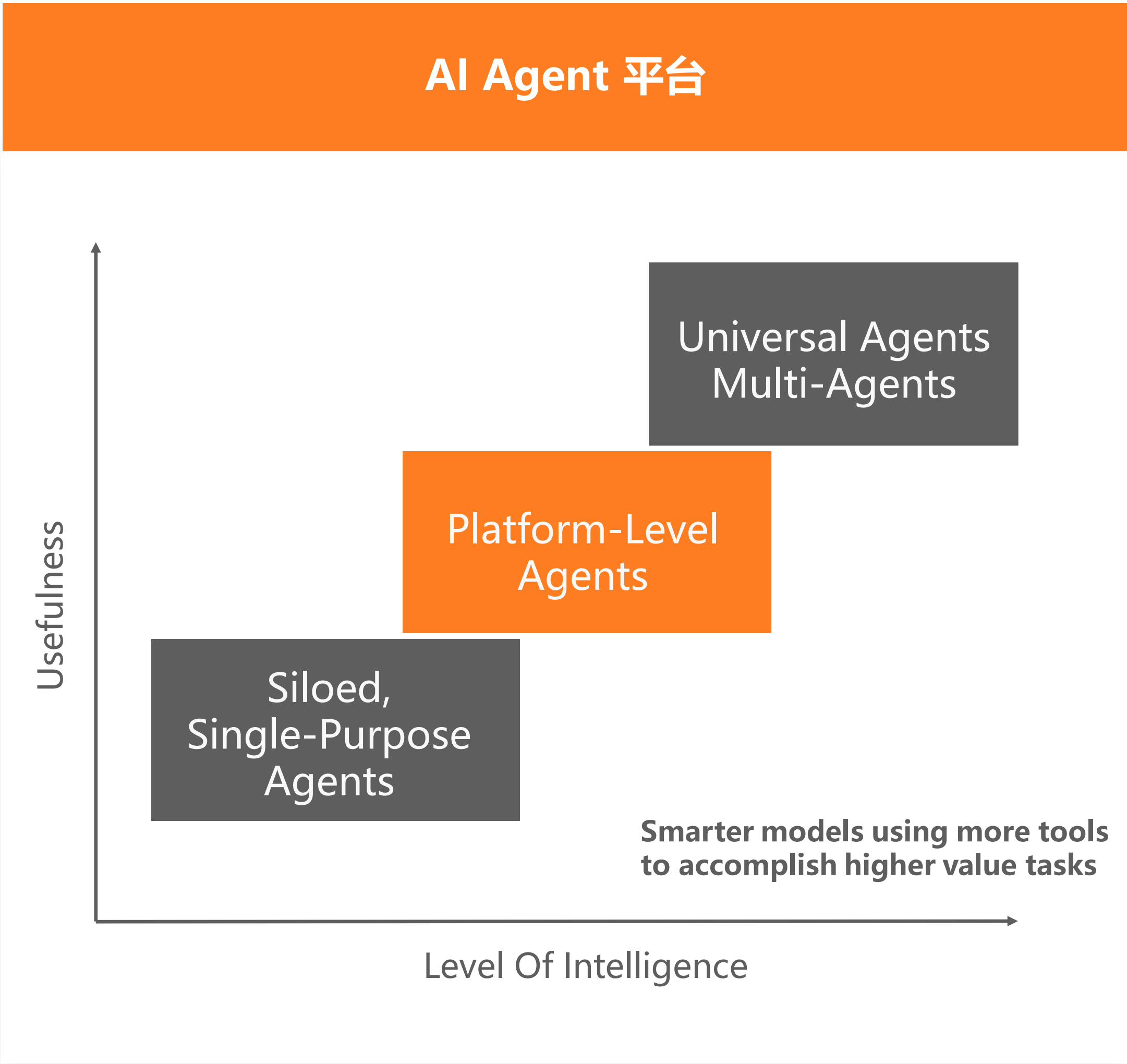
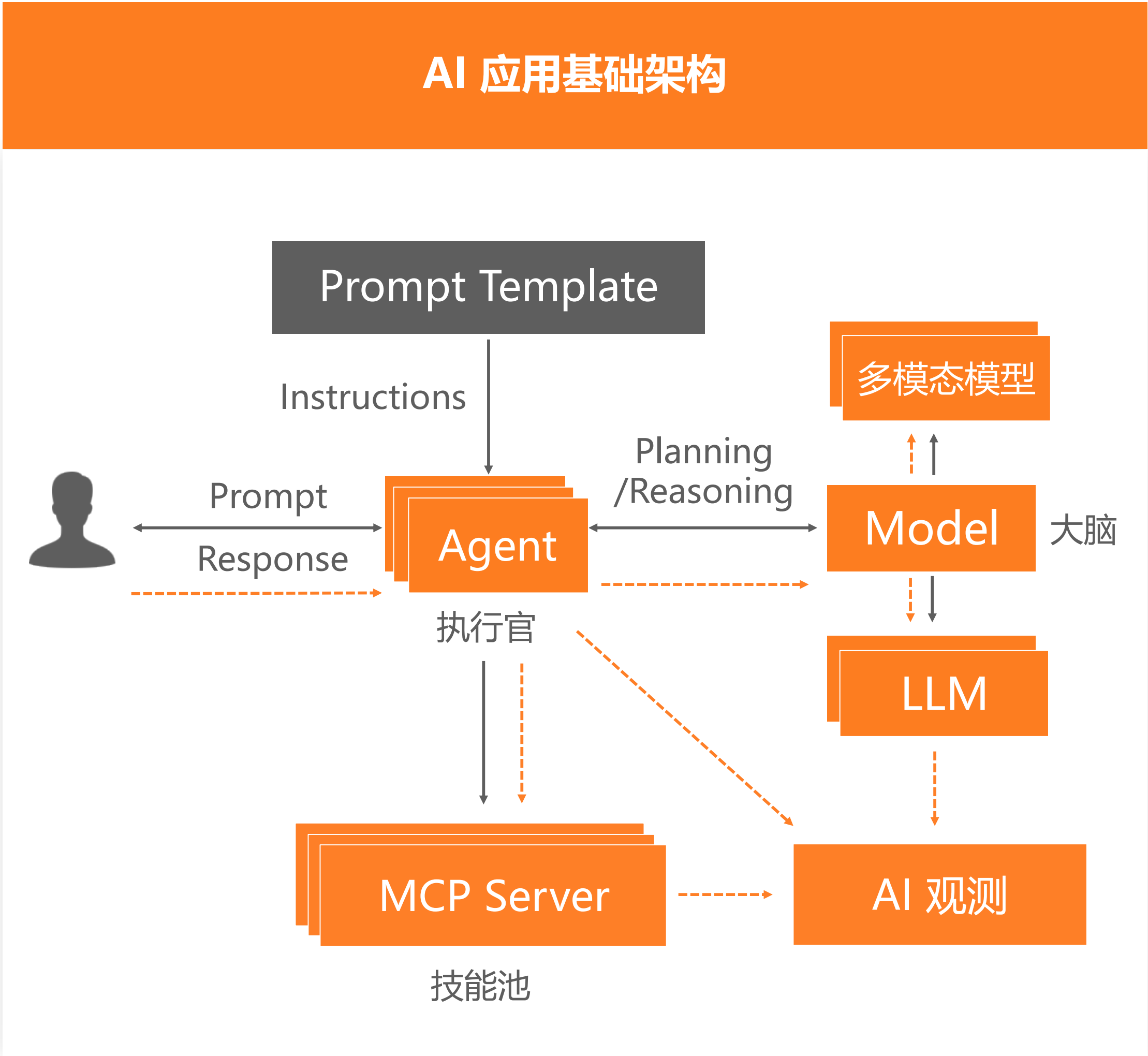
Existing Business Transformation



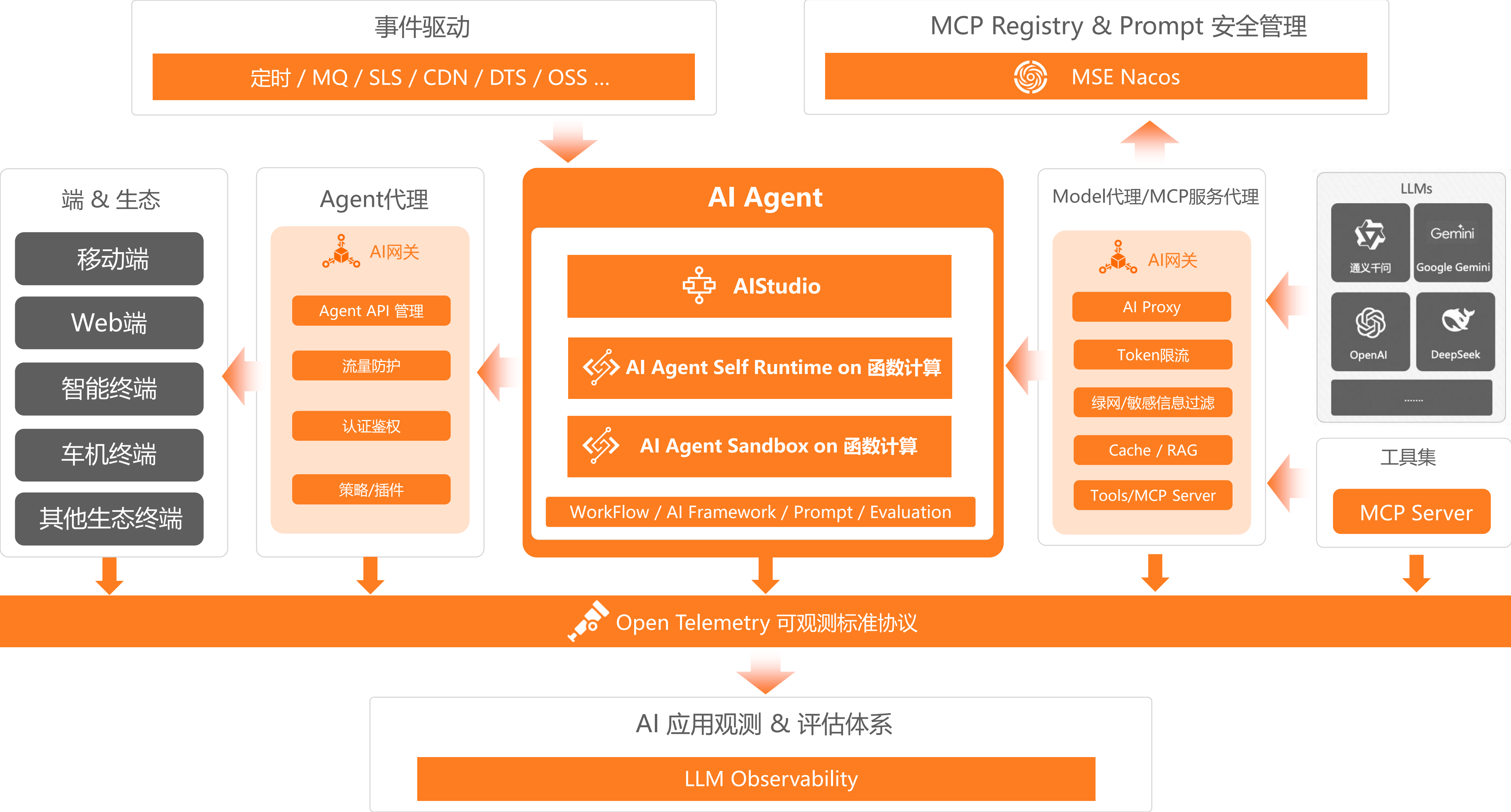
改造现有业务：为核心引擎注入AI动力

这是绝大多数企业会选择的路径。它指的是在企业现有的、成熟的核心业务系统（如ERP、CRM、SCM）中，嵌入AI Agent的能力，对其进行“智能化升级”。这种方式能直接作用于核心业务流程，价值释放路径更短、更明确。

AI 应用的核心是 AI Agent

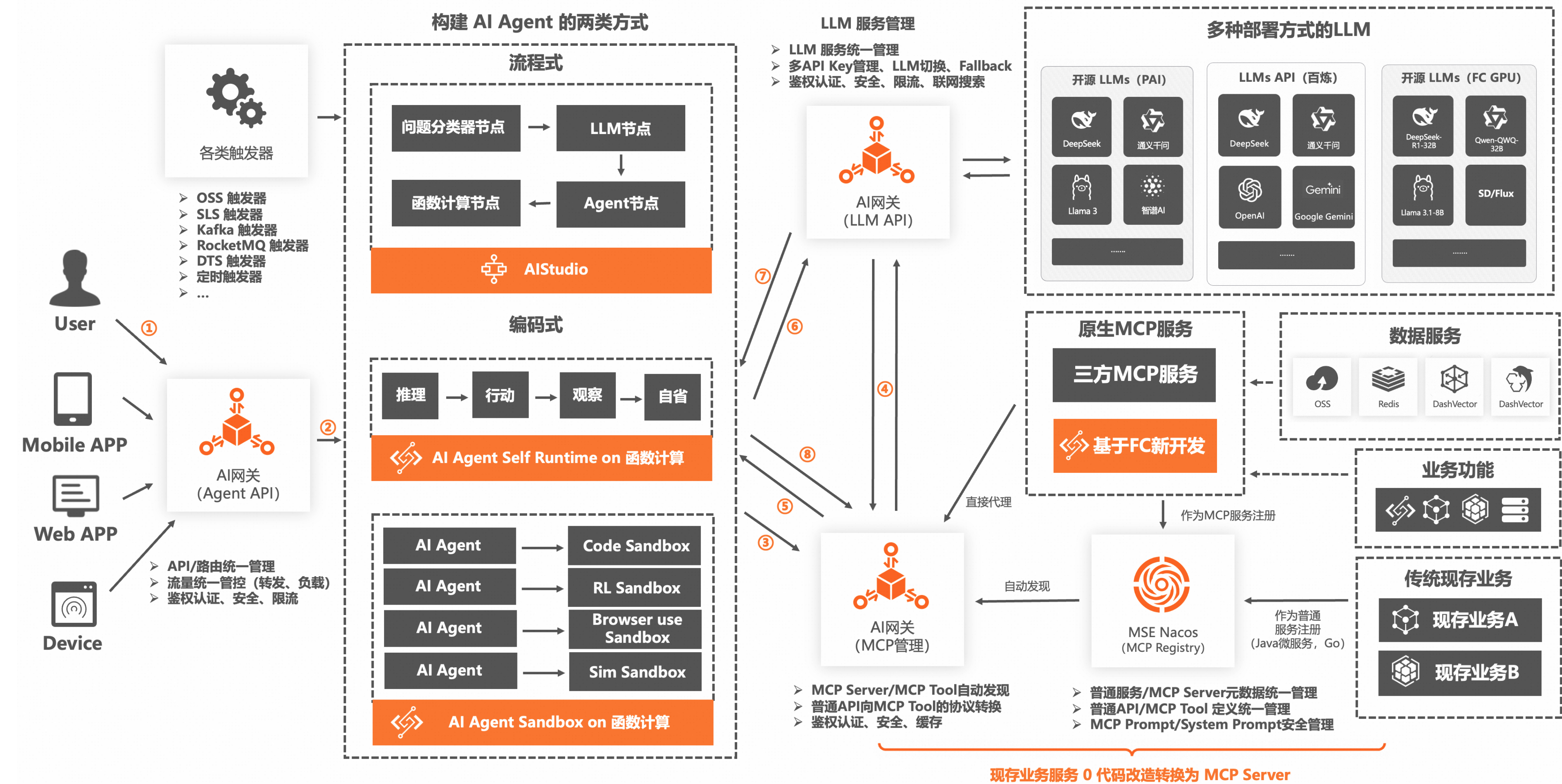


AI 应用架构新范式



AI 应用架构新范式创析

- 一个AI网关三种角色，具备统一的管控底座，同时又实现各角色的协同调度。
- MSE Nacos 发挥注册中心优势，增加MCP Registry能力，实现普通服务和MCP服务的统一管理，结合网关实现**现存业务0改造转换为MCP服务**。
- AIStudio为阿里云自研的低代码构建AI Agent的产品，解决开源Dify**高可用，稳定性，性能问题**，使AI Agent的运行引擎更稳定。
- FC具备丰富的触发器和各语言运行环境，基于Serverless计算自身的特性，完美适配**AI Agent自身运行环境**和**AI Agent Sandbox**的基础组件。



调用链路说明

- ① 用户向AI应用发起请求，请求流量进入AI网关，使用Agent API代理AI Agent。
- ② AI网关侧维护管理了不同类型的AI Agent的API或路由规则，将用户请求转发至对应的AI Agent。
- ③ AI Agent无论以哪种方式实现，只要它需要使用工具解决用户的问题，便向AI网关管理的MCP服务请求获取可用的MCP服务及工具的信息。
- ④ 因为AI网关处可能维护了很多MCP信息，可以借助LLM缩小MCP范围，减少Token消耗，所以可以通过AI网关代理的小参数LLM，做意图识别，进一步缩小MCP服务范围。（可选，需用户自己实现）
- ⑤ AI网关将确定好范围的MCP服务及工具的信息List返回给AI Agent。
- ⑥ AI Agent将用户的请求信息及从AI网关拿到的所有MCP信息再通过AI网关发送给LLM。
- ⑦ 经过LLM推理后，返回解决问题的一个或多个MCP服务和工具的信息。
- ⑧ AI Agent拿到确定的MCP服务和工具的信息后通过AI网关对该MCP工具做请求。

实际生产中 ③ - ⑧ 步会多次循环交互



02 AI Agent 概述

什么是 AI Agent



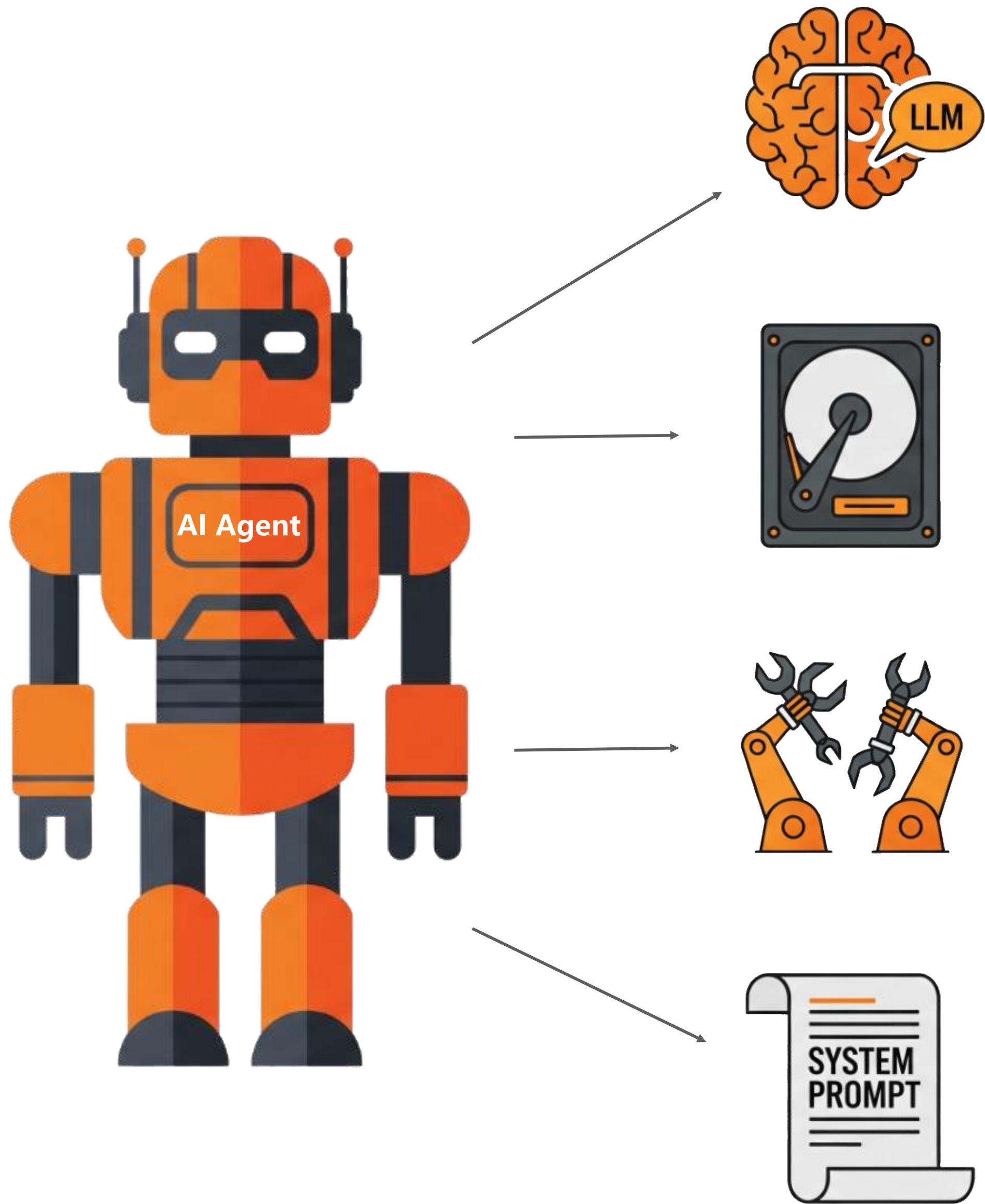
AI Agent 是一个系统

一个AI Agent其实是一个系统，包括以下三个核心内容：

- 使用大语言模型（LLM）来推理
- 可以通过工具执行各类行动。
- 执行思考（Think） -> 执行（Action） -> 自省（Observe） -> 纠错（既重复思考到自省的持续改进）这样一个循环。

AI Agent和Chatbot的最大区别是前者可以解决需要通过不同领域的知识和能力协同才可以解决的问题，通俗的说就是复合的、复杂的、多步骤的问题。

AI Agent 的核心组件



- 大脑，既大语言模型（LLM）
- 作用：识别自然语言，然后进行推理并做出决策。
 - 原则：选择最合适的大语言模型。（不同的大语言模型有自己擅长的领域和业务场景）
- 记忆，既存储服务（NoSQL或向量数据库实现）
- 作用：让Agent记得目标、偏好，以及过往的交互信息，从而实现多步骤执行，自省等能力。记忆里也分长期记忆和短期记忆。
- 手，既各类工具（MCP Server）
- 作用：为Agent提供外部能力，各类业务服务，数据库服务，存储服务等等。既执行LLM做出的决策。
- 指令，既系统提示词（System Prompt）
- 作用：定义Agent的目标和行为。

AI Agent 的推理模式 - ReAct 模式

自省 (Reflect)

评估是否需要继续推理->行动->观察以得到更趋近于用户目标的结果。

推理 (Reason)

使用LLM分析、理解上下文，明确用户任务目标。



观察 (Observe)

评估执行行动后得到的结果。

行动 (Act)

基于推理的结果，执行对应的行动。

AI Agent 的构建模式与 AI Agent 类型

编码式

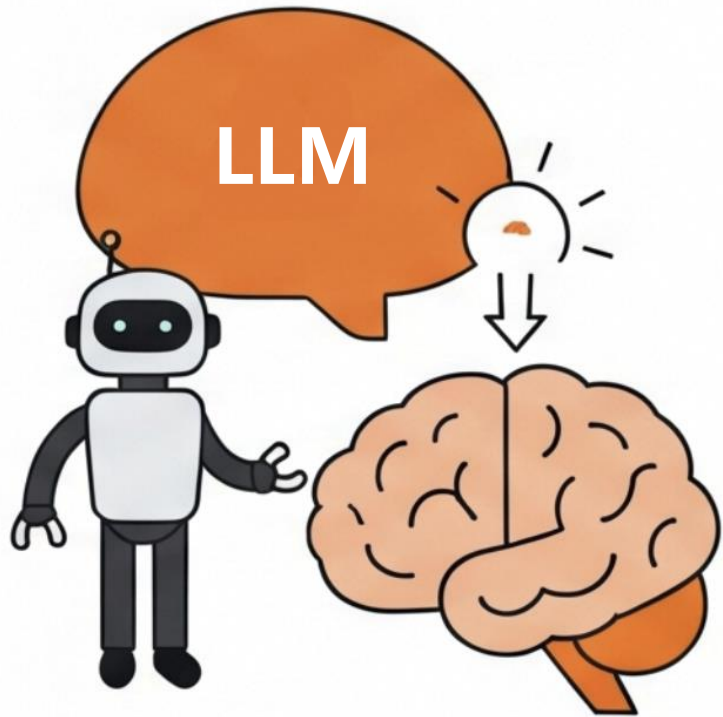


- Manual Coding
- LangChain
- LangGraph
- OpenAI Agents SDK
- Vertex AI Agents
- Crew AI
- Pydantic AI
- Spring AI Alibaba

低代码式

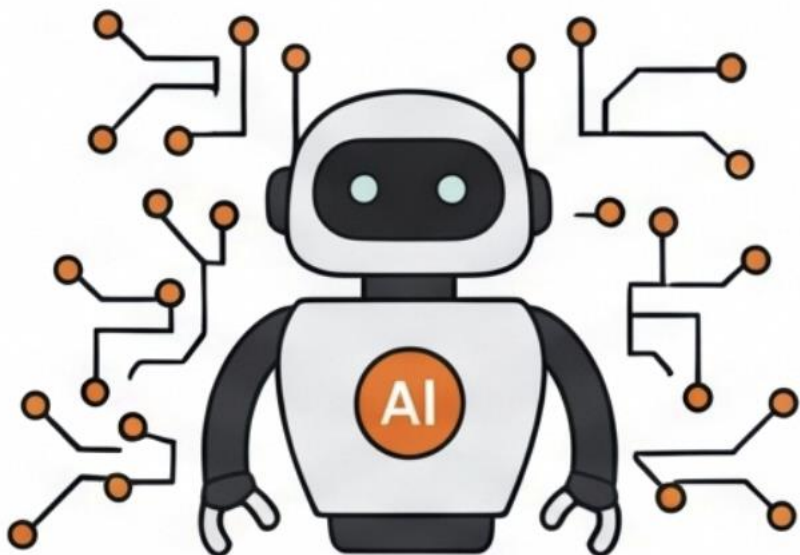


- 阿里云AIStudio
- 阿里云百炼
- Dify
- N8N



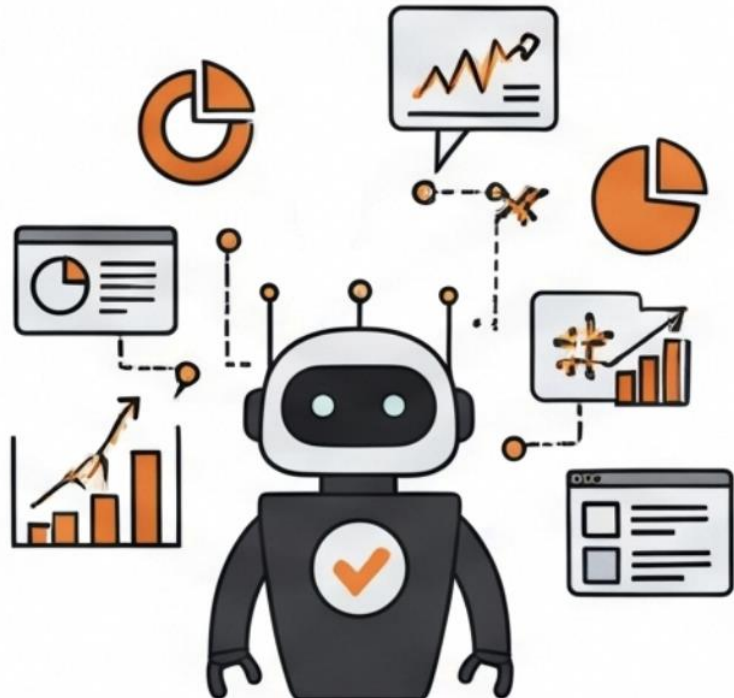
辅助基模（基础大语言模型）的AI Agent

当今基模的联网搜索、深度研究（DeepSearch）、编码能力都是需要AI Agent辅助的，这类AI Agent并不直接对用户透出。我们的实践中像Qwen3、智谱GLM等都属于这一类，通常都是做基模的公司会涉及到，并且以编码方式构建为主。



作为独立产品的AI Agent（通用AI Agent）

这类AI Agent大都还是基于主流的Chat模式，帮用户解答问题，规划任务等。我们的实践中像OpenManus、JManus、MiniMax Agent、昆仑万维等都属于这一类，通常都是做基模或者专门做通用AI Agent产品的公司会涉及到，并且以编码方式构建为主。



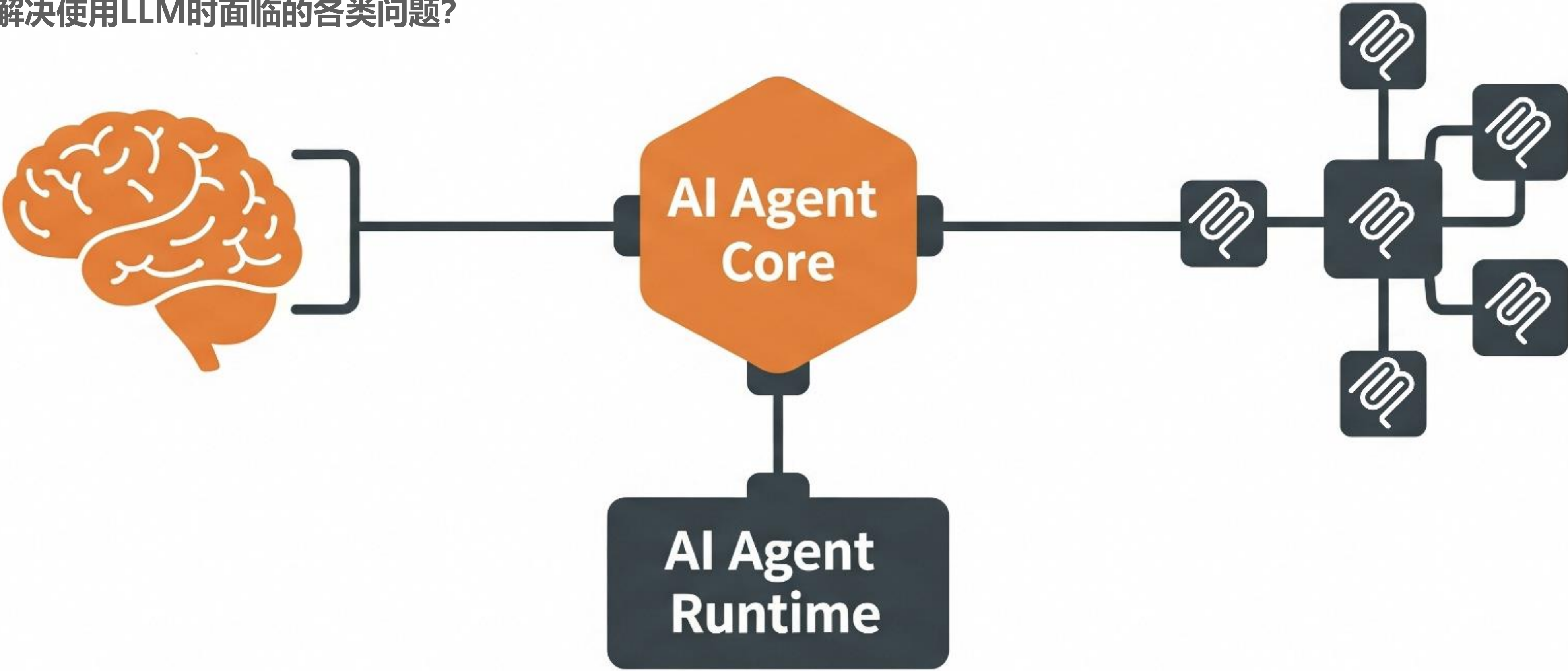
辅助现存业务的AI Agent

这类AI Agent就是目前广大互联网客户、泛企业客户期望构建或正在构建中的AI Agent，和客户自身的业务耦合比较紧密。我们的实践中像知乎、运满满、义乌小百货等都属于这一类，并且以低代码构建方式为主。

构建 AI Agent 的核心问题

如何解决使用LLM时面临的各类问题？

如何构建企业级MCP管理体系？



如何选择和使用AI Agent最合适的运行时？

03 函数计算FC概述

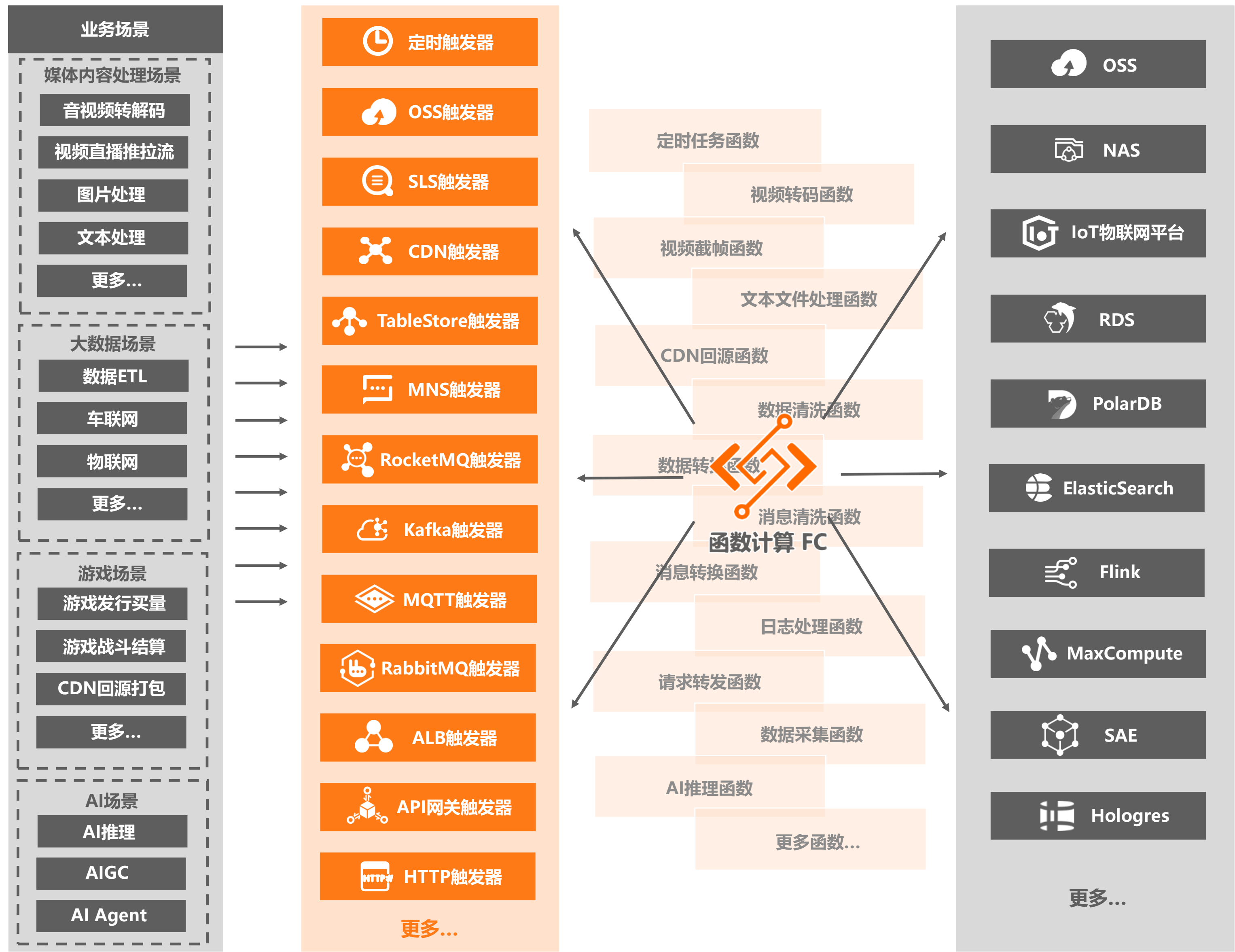
阿里云 函数计算FC 是什么

函数计算是事件驱动的全托管计算服务。使用函数计算，用户无需采购与管理服务器等基础设施，只需编写并上传代码。函数计算为客户提供了0.05C 128MB到16C 32GB不同规格的CPU实例和从1GB到48GB不同显存规格的GPU实例，用户可根据业务需求灵活选择资源规格。支持百毫秒级弹缩，近百种触发器类型满足各类业务场景，并且提供完善的日志、可观测、告警能力，是一款应用广泛的高可靠、高可用、高性价比的计算服务。



函数计算 FC 是云产品的连接器

函数计算FC提供了一种事件驱动的计算模型，函数的执行是由事件驱动的，函数的执行可以由多种方式（事件源）触发



函数计算 FC 触发器类型

- 按照触发器集成方式，函数计算支持的触发器分为以下三类：
- 双向集成触发器：您既可以在函数计算，又可以在事件源端配置触发器。
 - 单向集成触发器：目前只支持在事件源端配置触发器。
 - 云产品事件触发器：支持在函数计算配置触发器，同时支持在事件总线EventBridge创建函数触发规则，无需在事件源端配置。
- 详细信息可参考文档：https://help.aliyun.com/document_detail/53102.html

函数计算FC是云产品之间的粘合剂

函数计算 FC 提供了**20+核心云产品的原生触发器**，各触发器以白屏化的方式可以在函数计算控制台快捷配置，帮助用户快速的构建**多产品联动的自动化流程**。

通过函数计算触发器，可以实现拦截各云产品的行为，用户可以自定义函数对云产品的行为做处理，然后流转到了下游服务，从而实现不同产品之间的集成和联动。

客户案例

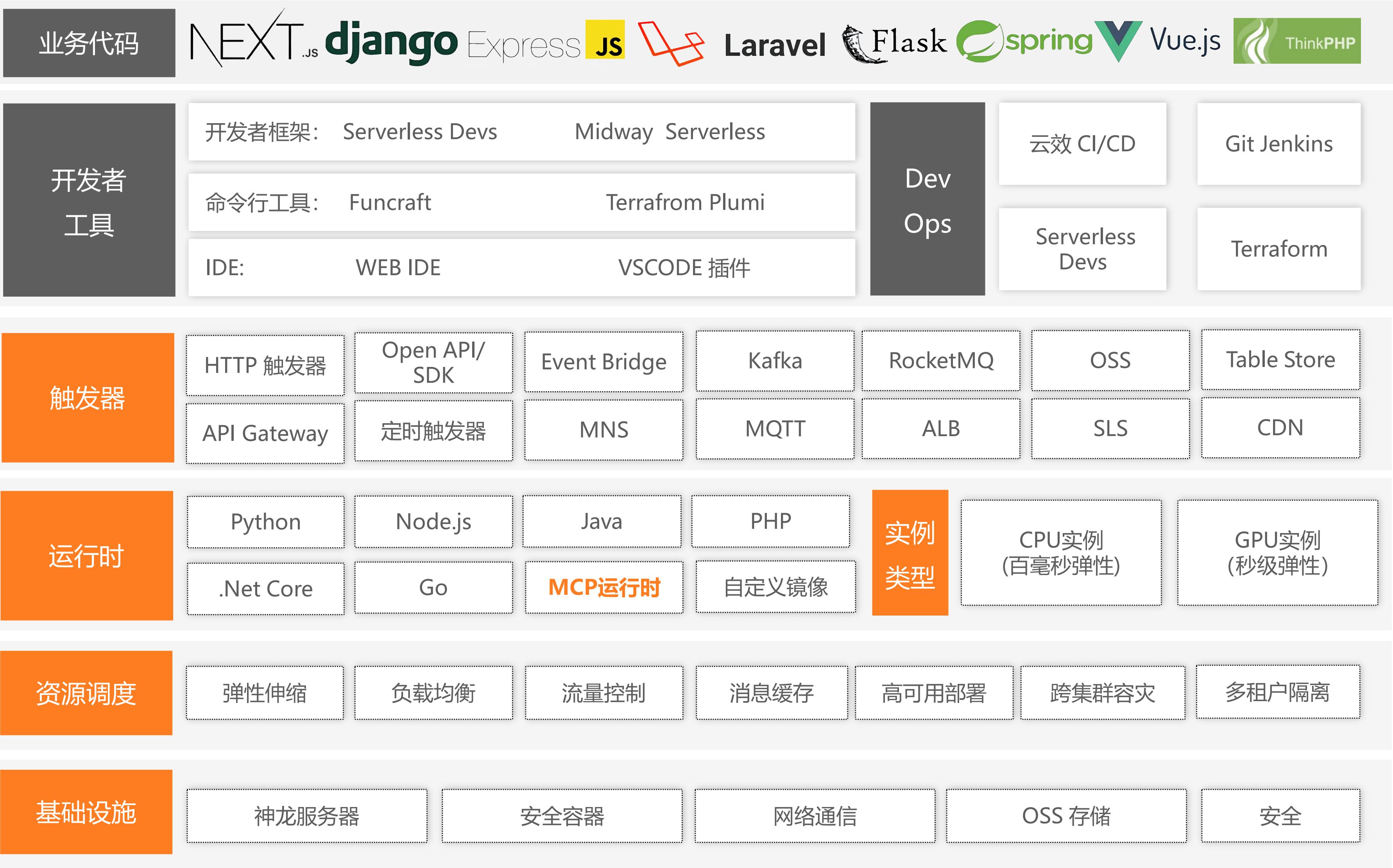
媒体内容处理场景	大数据场景
 新浪微博 weibo.com	 图森未来
 新东方 XDF.CN	 TOYOTA
 爱奇艺·体育	 HONDA
游戏场景	大并发场景
 心动	 高德地图
 环球新闻	 世纪联华 CenturyMart
 熊猫玩 xmwan.com	 1688

函数计算 FC 是云产品的连接器

函数计算

业务侧关注

平台侧提供



FunctionAI

- 模型服务
- AIStudio
- AI应用模板

任务编排

- CloudFlow

可观测

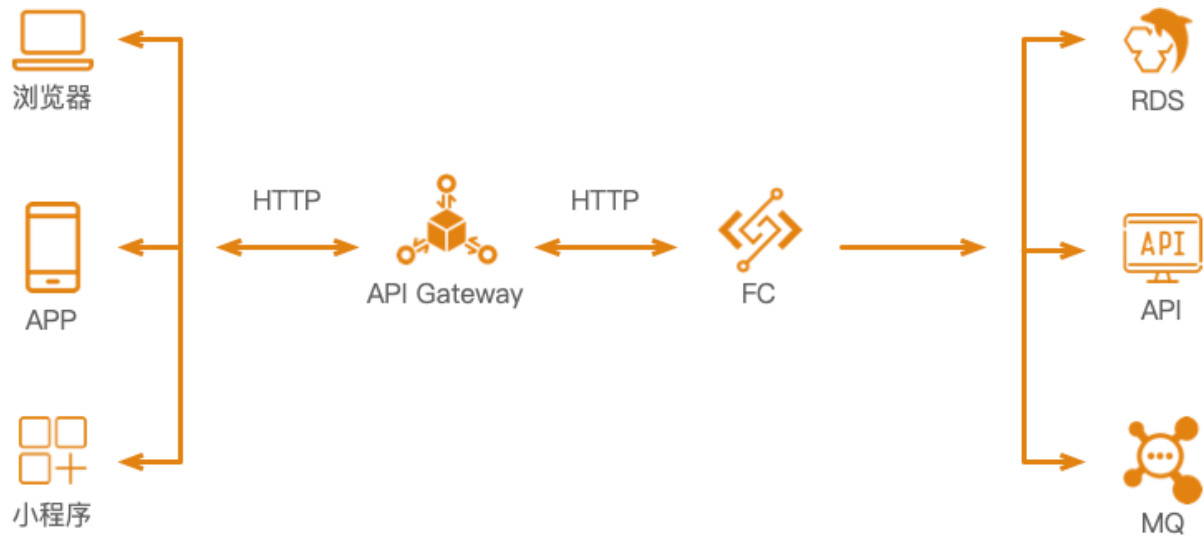
- 标准日志 (SLS)
- 监控告警 (云监控)
- 性能监控 (ARMS)
- 成本管家
- 操作审计

函数计算 FC 应用场景

HTTP应用通用方案

Serverless HTTP应用场景、Web应用场景

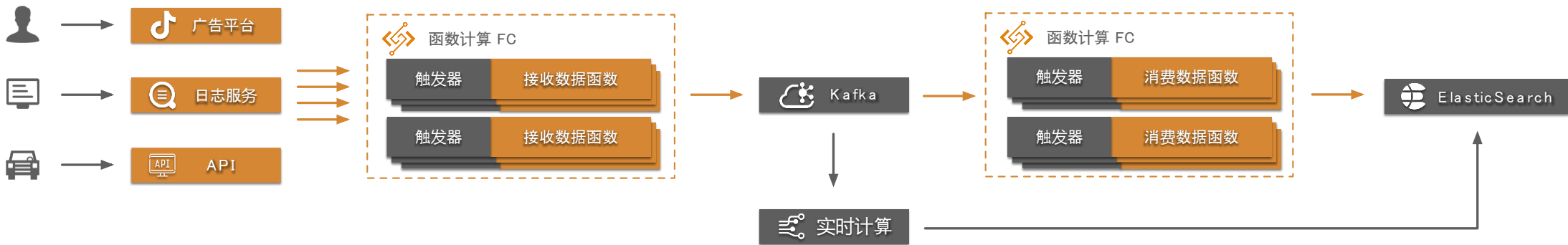
结合 API 网关，提供小程序，H5，WEB服务，REST API 等通用 HTTP 解决方案



大数据场景通用方案

Serverless 大数据ETL场景、车联网场景

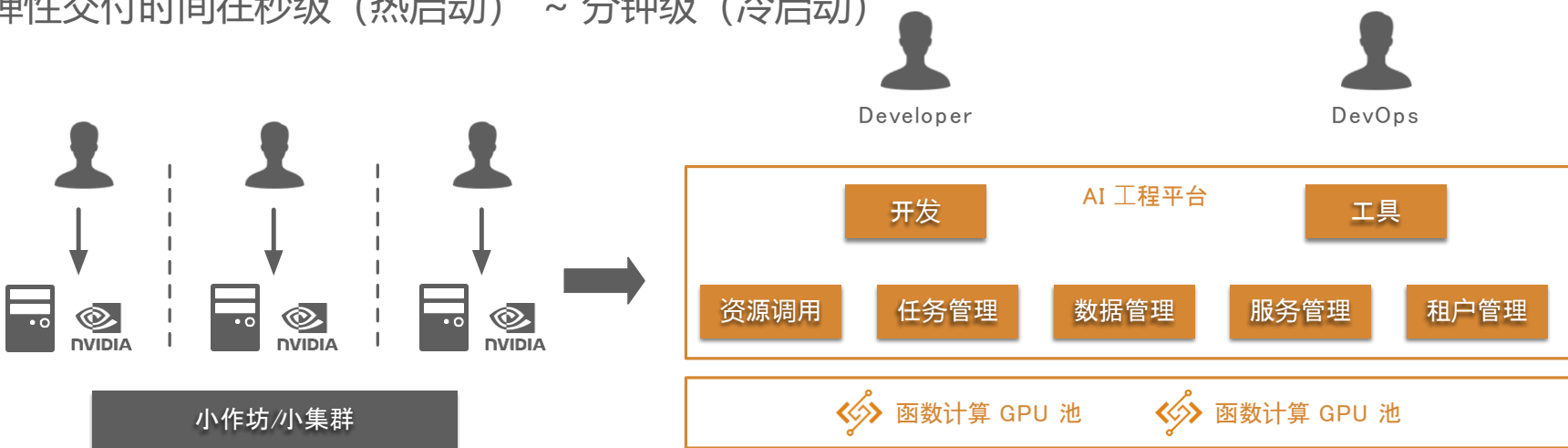
- 支持一键配置消息中间件数据消费，无需额外构建消费者，如 Kafka 消息备份、日志清洗、聊天消息分发等
- 支持自定义处理逻辑，承载多种多样的业务需求
- 支持高可用，支持配置错误重试



AI场景

Serverless GPU 场景

- 【在线/实时/离线】AI推理
- 弹性能力：提供最小1/24卡粒度的GPU虚拟化，将在线推理worker充分并行，单卡资源利用率最大可提高至8倍。，弹性交付时间在秒级（热启动） ~ 分钟级（冷启动）



内容处理通用方案

Serverless 音视频、图片、文本处理场景

- 支持一键配置对象存储事件监听，即可实时消费 OSS 事件，如自定义图片处理、ZIP文件解压缩、CDN自动刷新、数据实时同步等
- 自定义处理逻辑，支持多种编程语言，实现诸如文本分析、MapReduce等
- 毫秒级弹性伸缩，轻松应对负载的波峰波谷，实际案例支持10W+级QPS



Serverless游戏场景

Serverless 游戏场景

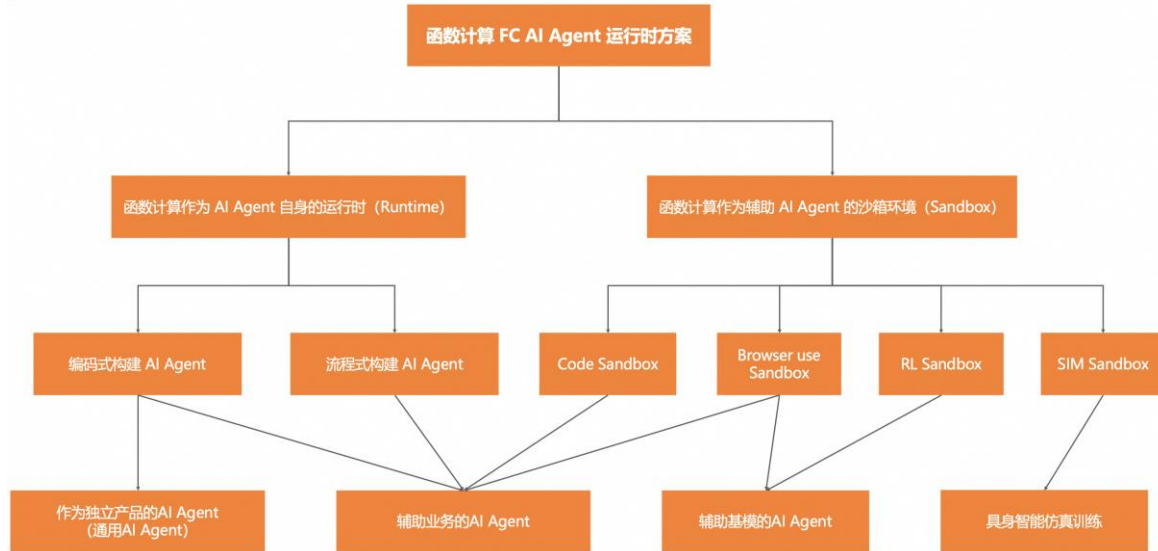
- 将游戏场景中通用的业务能力 进行抽象，从游戏主服中进行剥离，沉淀到统一服务层，最大化的提升资源利用率，减少系统维护工作量和成本
- 函数计算接管强 CPU 密集型的战斗结算场景，稳定应对开服、运营活动时的流量脉冲



AI Agent场景

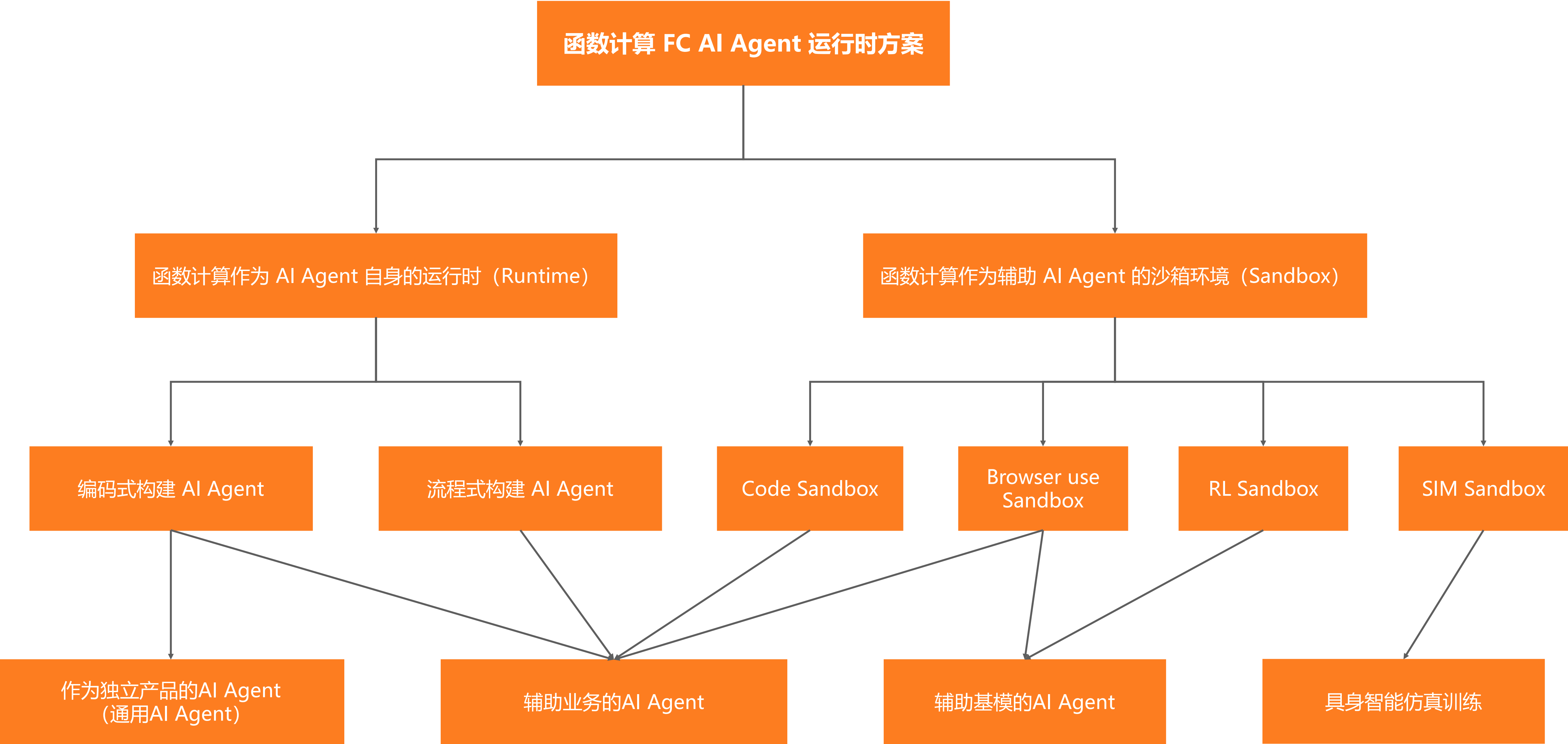
Serverless AI Agent 场景

- 作为AI Agent Code Sandbox
- 作为AI Agent RL Sandbox
- 作为仿真训练 Sandbox
- 作为Agent Runtime
- 支持所有主流开发语言运行环境
- 托管SD, ComfyUI, Isaac Sim, SWE-Bench 等综合框架
- 具备可视化构建Agent，MCP服务能力

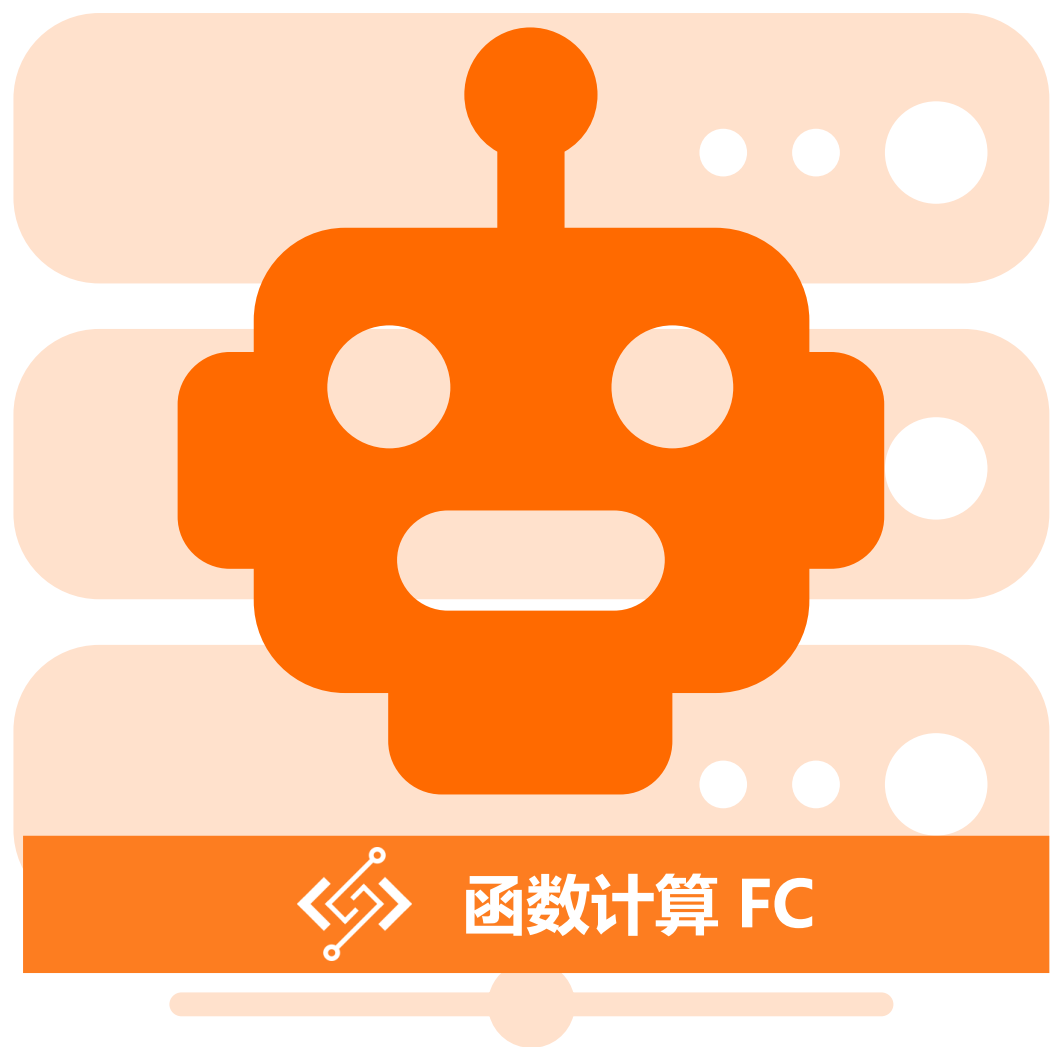


04 函数计算FC作为AI Agent运行时

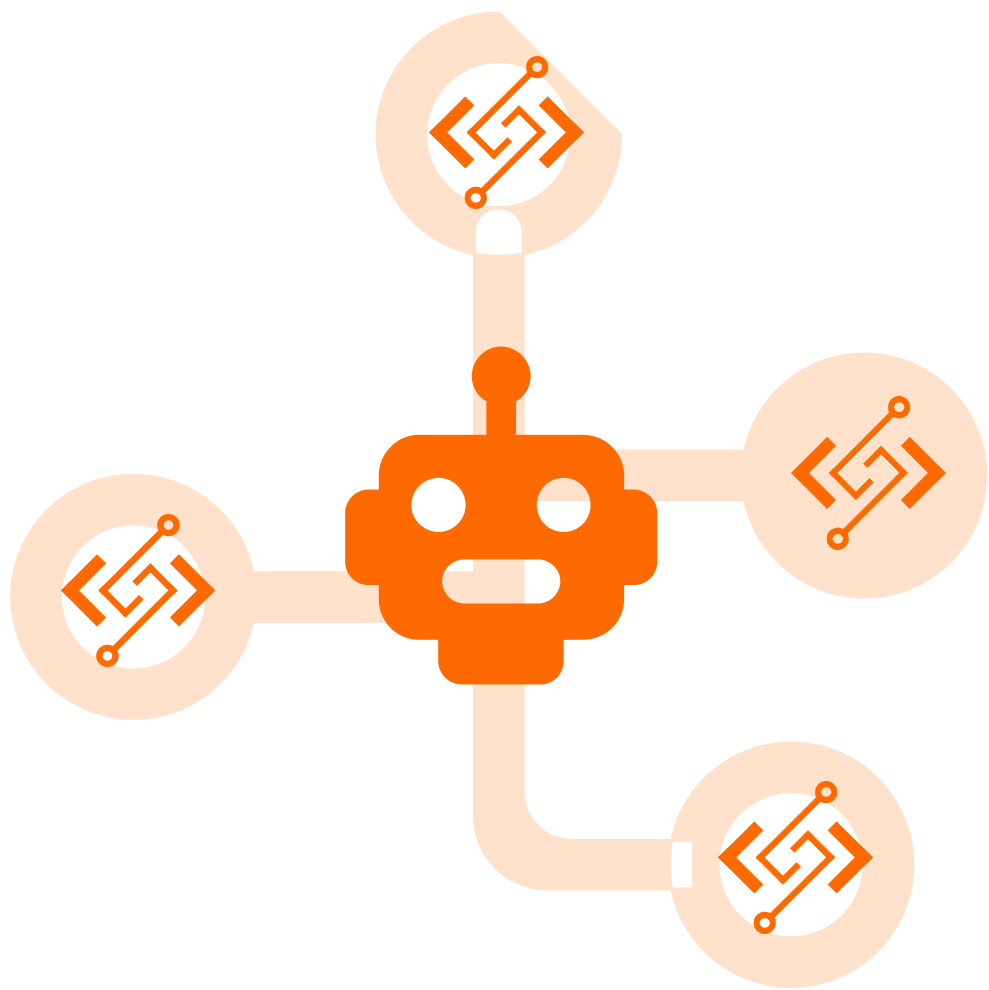
函数计算FC支撑AI Agent方案拓扑图



函数计算支撑 AI Agent 的两种方式



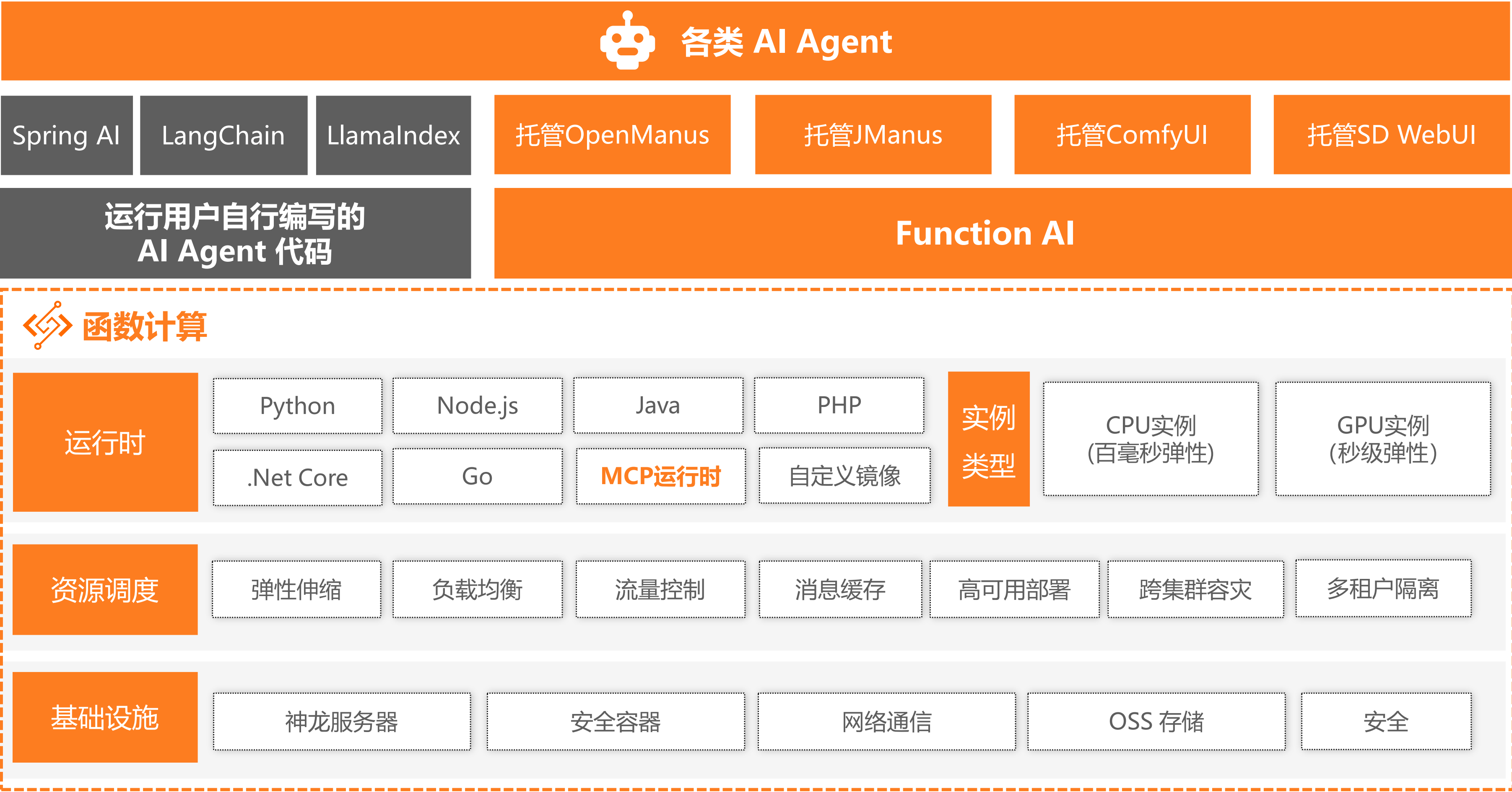
函数计算作为 AI Agent 自身的 Runtime



函数计算作为辅助 AI Agent 的 Sandbox

函数计算作为计算资源运行AI Agent – 编码式

函数计算FC作为计算资源提供细粒度、高弹性、强安全的AI Agent运行环境。



两种类型

运行在函数计算FC上的AI Agent有两种类型：

- 用户自行编写的，自研的AI Agent。或者如使用Spring AI Alibaba、LangChain、LlamaIndex等开发Agent的综合框架。
- 在FunctionAI平台上，已经托管了一些现成的AI Agent组件，比如OpenManus，Jmanus，ComfyUI，SD WebUI等。

优势和特点

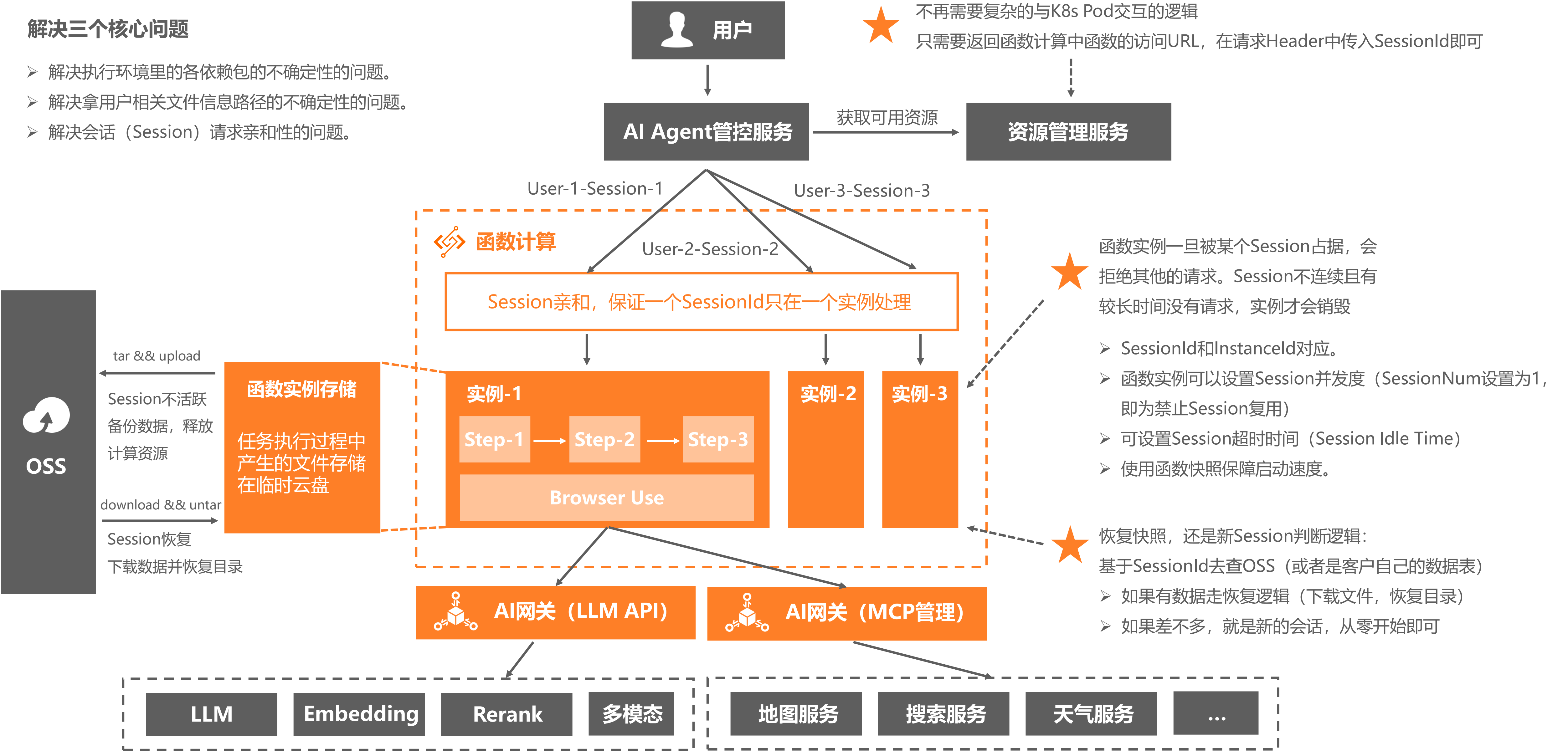
- 函数计算 FC 触发器机制，实现 AI Agent 可灵活被调度。
- 函数计算 FC 按请求扩缩，提升AI Agent 资源利用率，降低资源成本。
- 函数计算 FC 动态挂载存储机制，提升AI Agent 业务逻辑灵活性。
- 函数计算 FC 函数实例动态安装依赖包，提升 AI Agent 业务形态多样性。
- 函数计算 FC 和多个云上产品做好了集成，降低运维成本。

函数计算构建 Chat AI Agent



解决三个核心问题

- 解决执行环境里的各依赖包的不确定性的问题。
- 解决拿用户相关文件信息路径的不确定性的问题。
- 解决会话（Session）请求亲和性的问题。



函数计算 FC + CloudFlow 可视化构建 AI Agent



类Dify
可视化流程



自研
流程引擎

AI Studio	
Function AI	
函数计算 FC	CloudFlow
易用的同时性能更强	优势和特点
兼容Dify的流程构建习惯 ➢ 使用Dify可视化流程编辑器的设计语言和UE，最大限度兼容用户在构建流程时的习惯。 具备正统流程引擎的高性能 ➢ 基于函数计算FC和云工作流CloudFlow实现的生产级流程引擎。	➢ 支持函数计算节点，使构建流程的灵活性得到大幅度提升。 ➢ 默认支持最大1000QPS，且可以按需继续提升。 ➢ 多节点复杂流程依然具备稳定高可靠的执行性能。 ➢ 除了HTTP以外，还支持多种调度方案，比如OSS，SLS，Kafka，RocketMQ等。 ➢ 具备完善的可观测能力，包括整体流程和具体的每个节点的可观测。

AIStudio 控制台



AIStudio是阿里云自研的可视化构建AI Agent的产品。底层的工作流引擎基于阿里云2018年就商业化的产品云工作流（CloudFlow）， 底层算力基于函数计算。而前端的可视化部分我们基本沿用的Dify的设计语言。目的很简单：让用户不改变使用习惯的前提下享受到更灵活、更稳定、性能更好的可视化构建AI Agent的产品。

FunctionAI / 项目列表 / 项目 test-aistudio

返回函数计算控制台 钉钉答疑群 帮助文档

← test-aistudio

全量部署 创建模板 项目配置 费用概览 删除

新建服务 搜索

AI-Flow

流程编排 已部署

orders

Web 服务 已部署

express

Web 服务 已部署

competent-gould

流程编排 待部署

distracted-torvalds

模型服务 待部署

frosty-lalande

模型服务 待部署

focused-fermi

流程编排 待部署

AI-Flow

地域 华东1（杭州） cn-hangzhou ARN 快速模式

工作流描述 - 流程模式

角色名称

编辑 部署 删除

流程定义 执行记录 流程调度 监控 部署记录

所有和LLM交互的节点，都是通过配置AI网关代理的地址与LLM做交互，可以实现保持域名不变，只切换LLM的Model Name即可切换不同平台的模型

查询订单使用工具实现，工具底层是基于函数计算实现的服务，配置到FunctionAI的工具中

该Agent配置的MCP服务是通过AI网关将普通服务转换而成的

使用函数计算节点，引入了检索百炼知识库的函数

开始

question 必填

模型选择

qwen-max

分类 1 order_status

分类 2 return_policy

查询订单或物流信息

qwen-turbo

分类 1 order_info

分类 2 express_info

归纳问题

qwen-max

提取订单ID

qwen-turbo

ORDERS

查询订单结束

AGENT-查询物流信息

MCP FunctionCalling

qwen-turbo

查询物流信息结束

百炼知识库检索

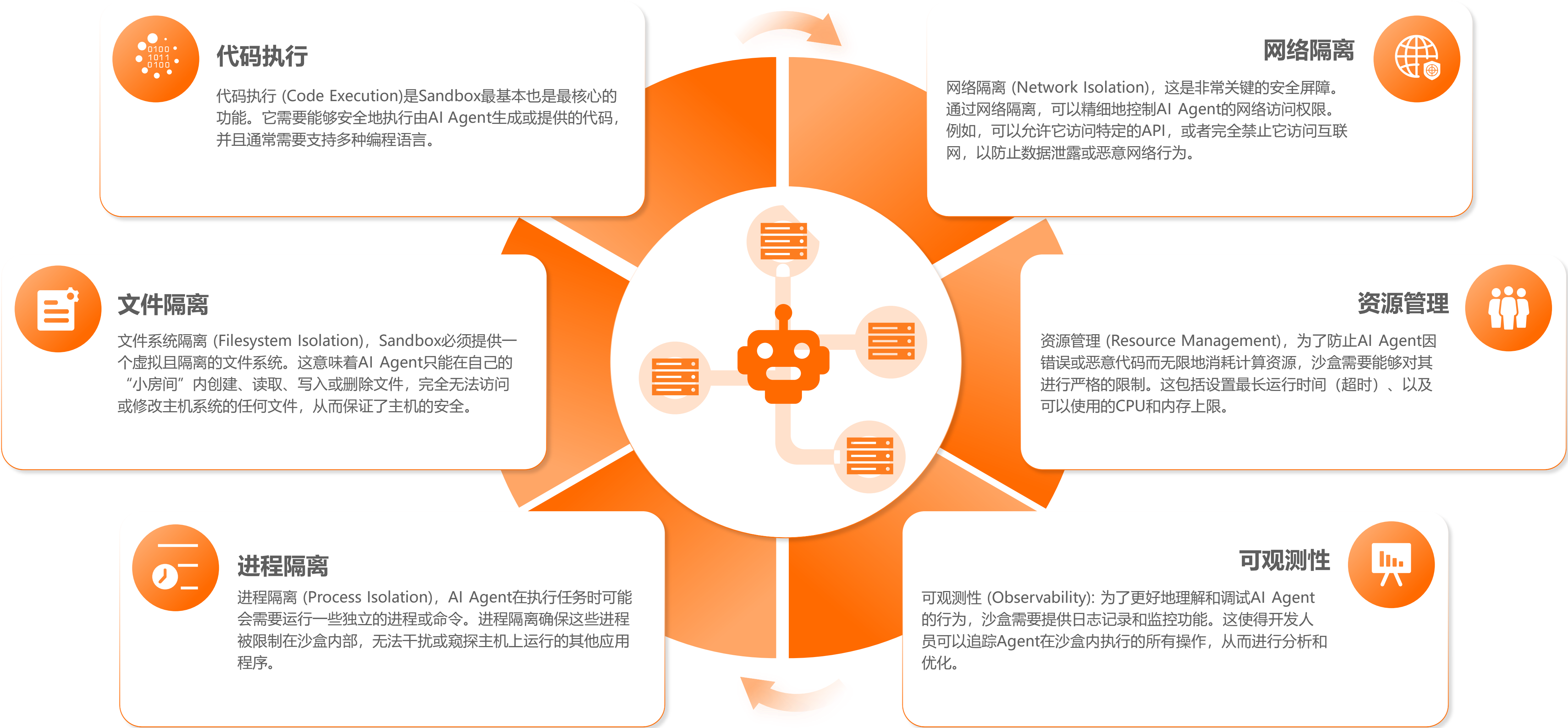
SYNC bailian_repo_retrieval / LAT EST

退换货流程结束

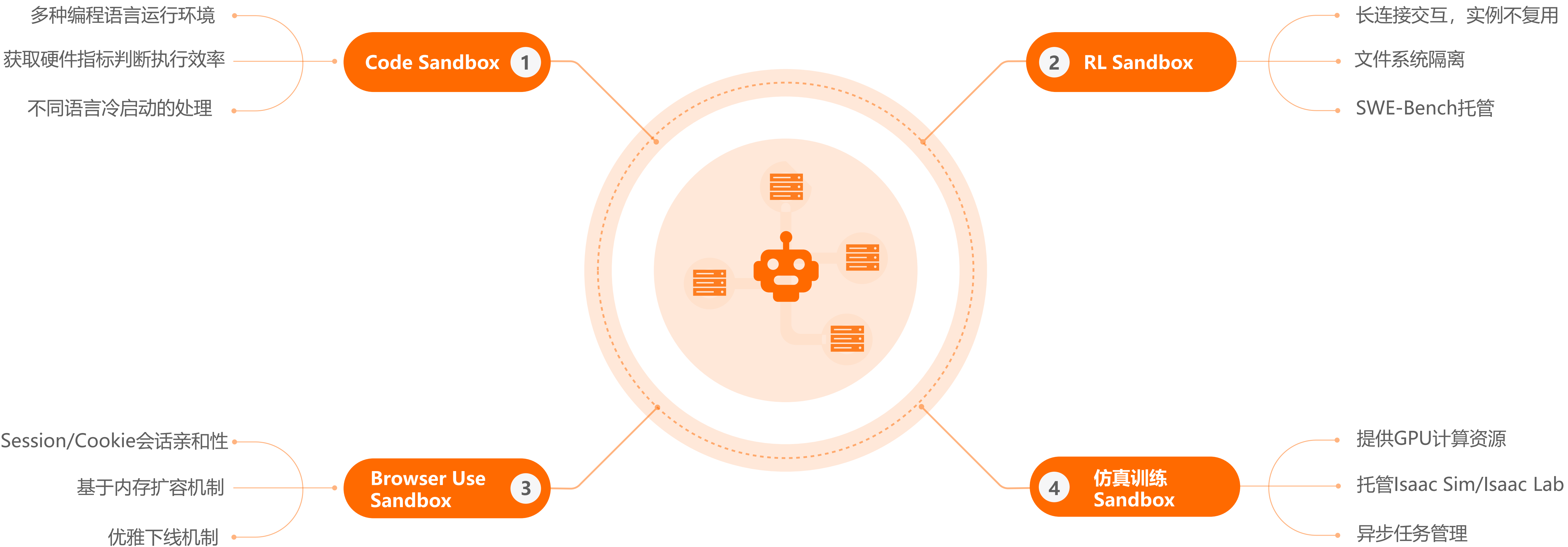
05 函数计算FC作为AI Agent Sandbox

AI Agent 为什么需要 Sandbox

为了确保AI Agent能够安全、可控地运行，一个强大的沙盒环境至关重要。这就像是为AI Agent提供一个安全的游乐场，让它在其中探索和执行任务，同时又不会对外部真实世界造成意外影响。



AI Agent 的 Sandbox 场景

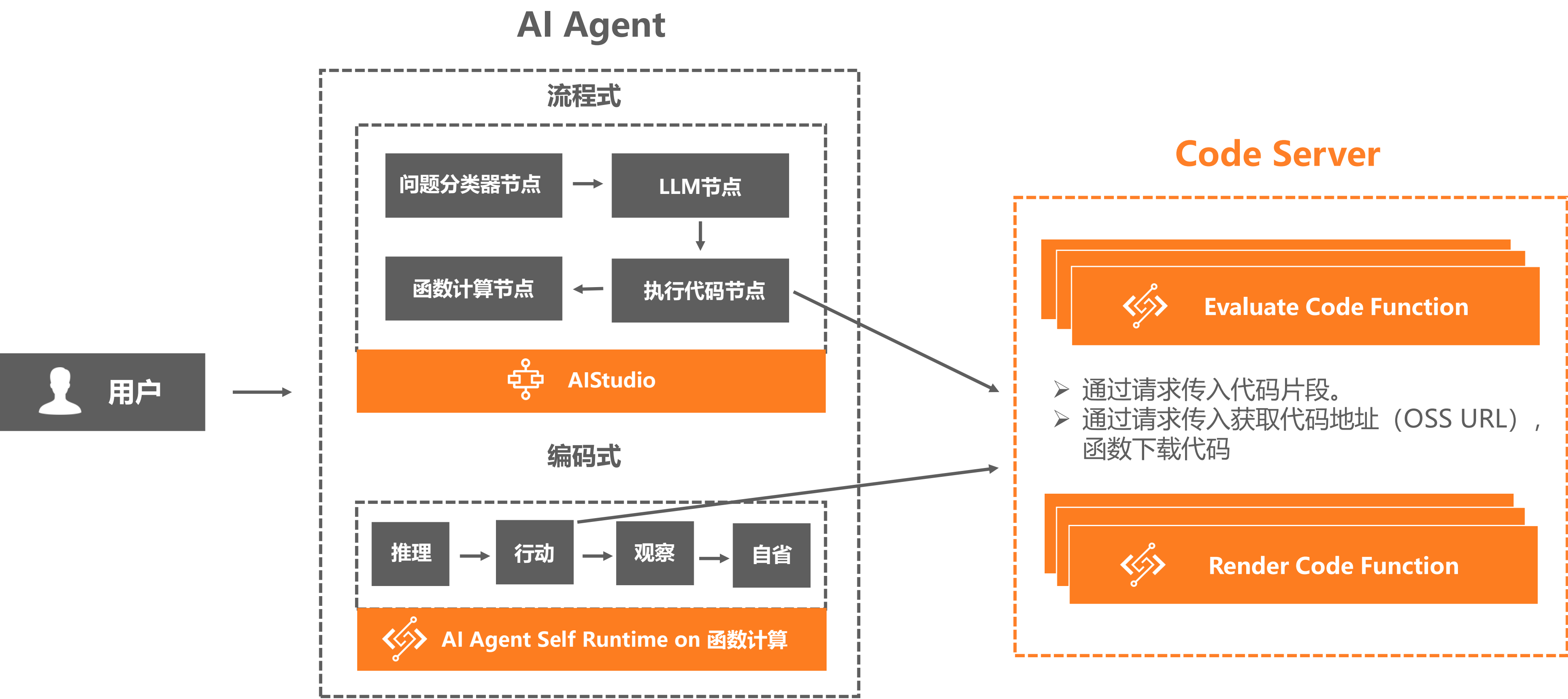


通用能力： 任务流程编排能力，多种被调度能力，实例快速拉起能力，与云产品的集成打通能力

Code Sandbox

这一类场景的本质就是在一个隔离的环境中运行由用户生成的或者LLM生成的代码，分为两种业务场景：

- 协助训练基模的Coding能力：给LLM喂需求，由LLM生成代码，然后拉起函数计算FC实例，运行代码，评判结果。
- 实时运行展示用户编码类的任务：这里包括执行后端代码，也包括执行渲染前端代码。无论是基模公司还是互联网客户的AI场景，都有相似的需求。比如Gemni的Canvas能力，千问的网页开发能力，MiniMax的Agent生成代码并运行的能力等。



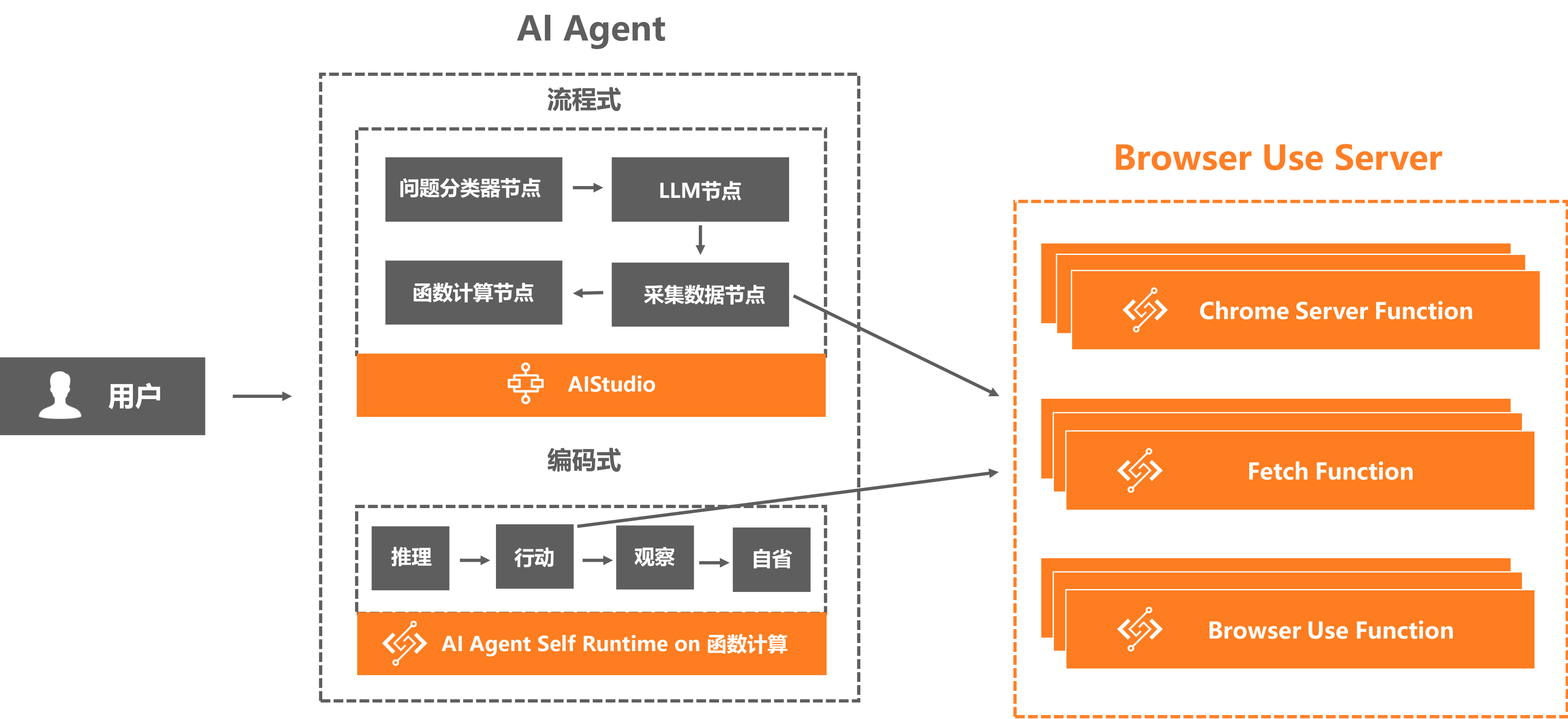
Code Sandbox on FC 优势

- 函数计算FC具备所有开发语言运行环境，适应各行各业，各类运行代码的需求。
- 函数计算FC提供能把这些数据拿走的能力（底层提供8代机，具备获取这些指标的接口），数据抓出来后给到另一个服务做衡量（另一个函数）。
- 函数计算FC支持实例内再起多线程执行子任务的能力，得益于函数计算FC的实例是完全独立的环境，只要函数规格够，多线程运行也不会影响其他实例，不会产生资源争抢。
- 对时延要求高且非常敏感的场景，广告领域的RTA绝对算一个，函数计算FC有成熟的RTA方案，并且支撑着不少大客户的RTA业务，所以在优化冷启动，解决时延方面有足够的经验。那么在Code Sandbox这个场景通常会使用弹性实例与毫秒级快照实例组合的方式来保证时延要求。

Browser Use Sandbox

在AI场景下，当前Browser Use主要有两类主要的应用场景：

- 辅助数据采集，比如需要登录的一些网站，获取论文报告等。
- 做联网搜索，目前主流搜索引擎的API能力参差不齐，且价格不菲，所以通过Browser Use做联网搜索在灵活性和成本方面都是较优的选择。

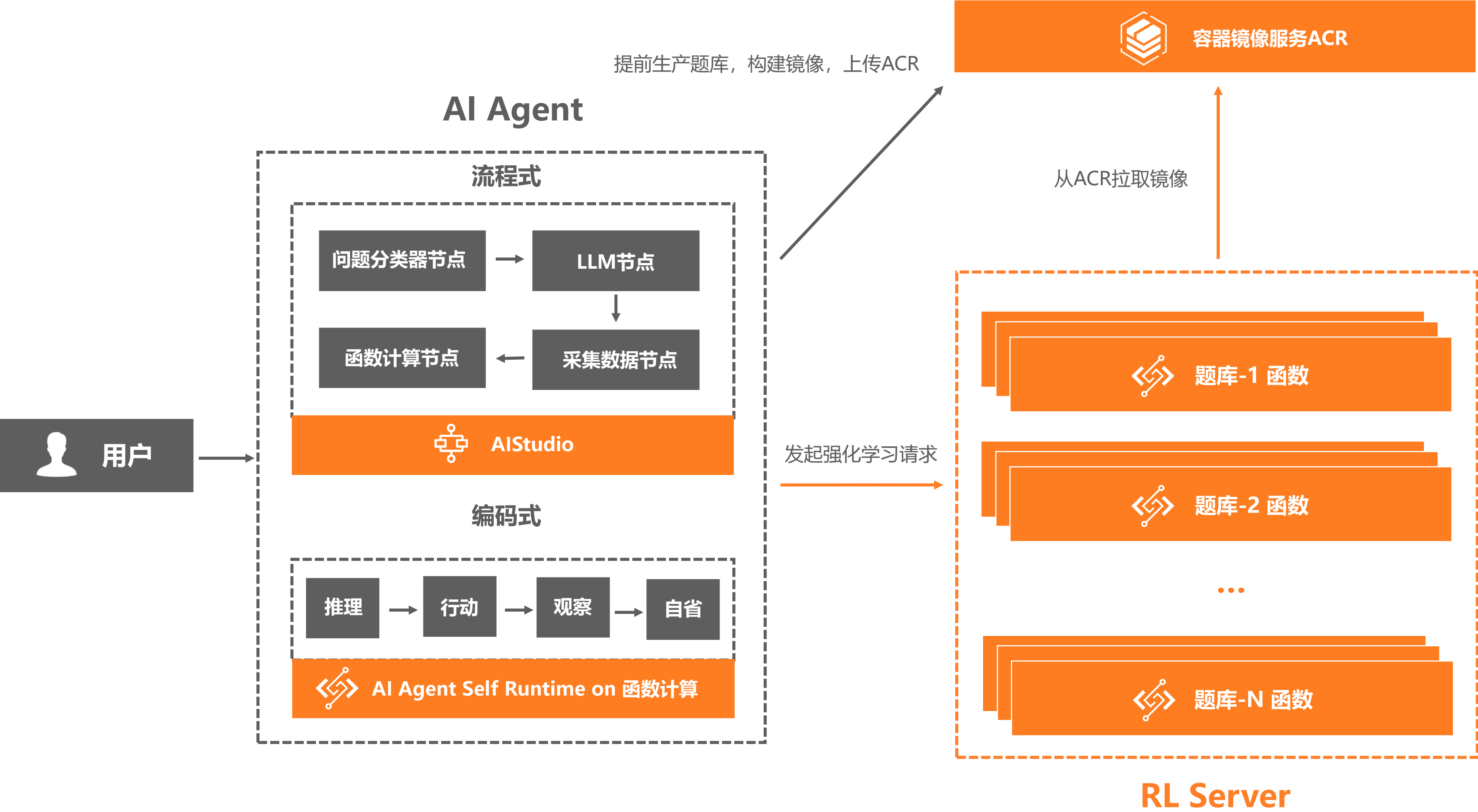


Browser Use Sandbox on FC 优势

- 需要Session/Cookie亲和性。辅助数据采集时，需要登录后才能获取到数据，所以需要相同Session的请求分配到同一个实例中，避免反复登录。
 - 函数计算FC支持会话（Session）亲和性的。所以也是天然适配Browser Use的特性。
- 需要基于内存扩容，这个场景比较吃内存，且大多数语言内存回收机制都不好。
 - 函数计算FC默认按请求扩容，此外还支持用户配置按时间和并发度扩容，为了支持Browser Use Sandbox场景，又支持了按内存扩容的能力。
- 优雅下线，也就是实例要销毁时做Browser Use操作的后处理。
 - 依托函数计算FC的生命周期管理能力，通过prestop钩子函数做Browser Use收集数据的后处理操作。

RL Sandbox

有一些基模客户或做通用AI Agent的客户，会专注在垂直类场景，这类客户会针对特定场景对LLM或AI Agent算法做定向强化学习。



RL Sandbox on FC 优势

➤ 安全性：Agent在训练初期的行为往往是随机且不可预测的。在沙箱中，错误的决策不会造成任何实际损失。

➤ 高效率与可复现性：沙箱环境可快速拉起，快速复制相同的环境，让Agent在短时间内经历海量的训练。同时，研发可以精确控制每个环境的每一个变量，从而能够复现实验结果，进行可靠的对比分析。

➤ 降低成本：不希望过多维护IaaS资源，随用随拉起，并且强化学习也不是实时业务，如何最大限度提升资源利用率也是降低成本的优化手段。

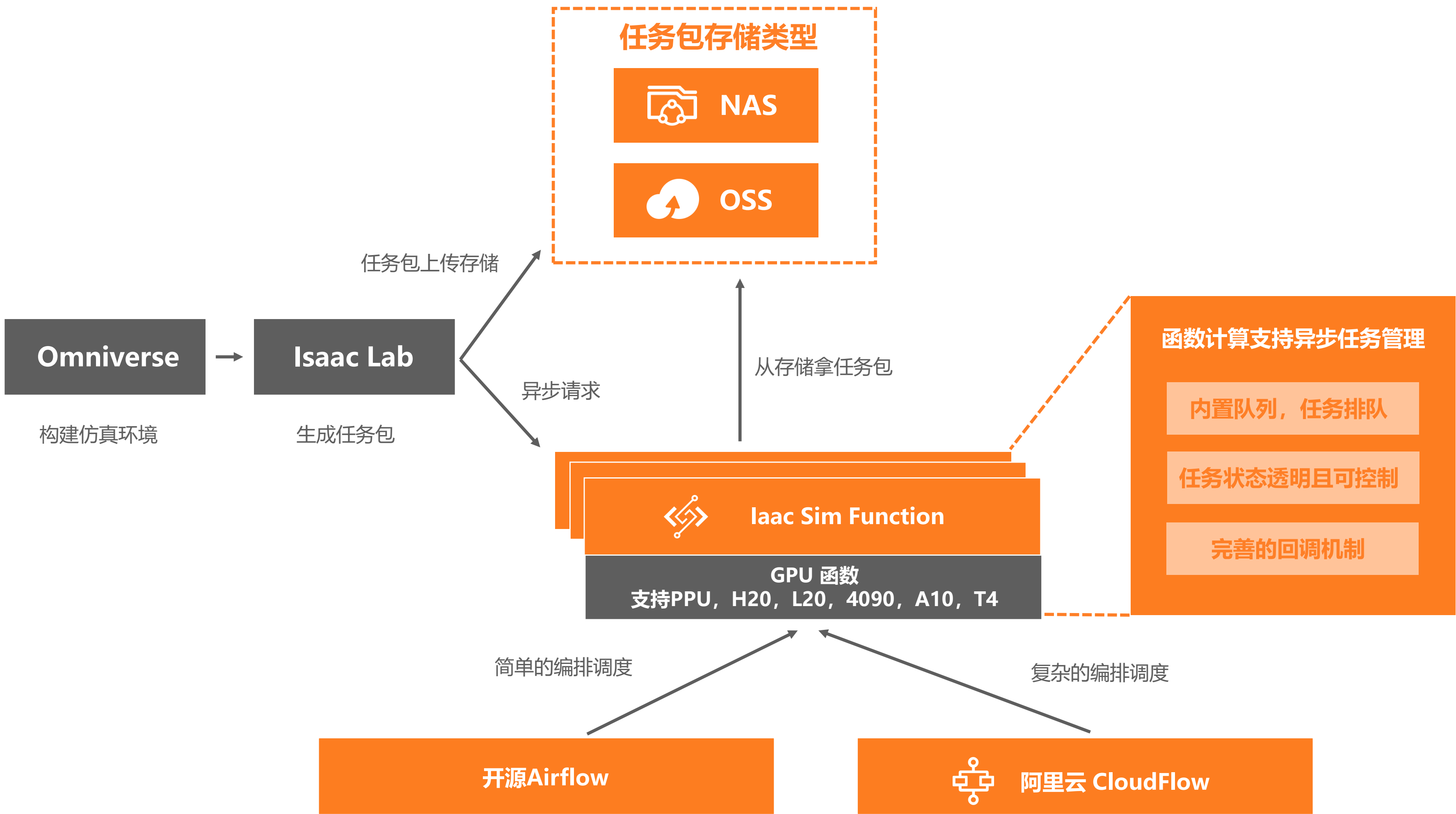
➤ 运行环境完整性：沙箱环境不要有太多限制和约束，期望和一台Linux机器一样去使用。甚至可以设置一些系统级参数。

Sim Sandbox



仿真训练Sandbox场景目前主要聚焦在具身智能场景。具身智能仿真训练基本流程：

- 使用NV Omniverse提供的可视化界面，构建虚拟环境，准备环境数据。
- 构建好仿真环境和数据后，生成任务包，将任务包分发到GPU服务跑训练任务。该服务使用的框架大多数也是NV Omniverse里的Isaac Sim。分发任务的逻辑通常会使用Airflow，且Airflow的流程是比较简单的。
- GPU服务跑完训练任务后，状态会回调Airflow，由Airflow统一来展示这次任务的执行结果。

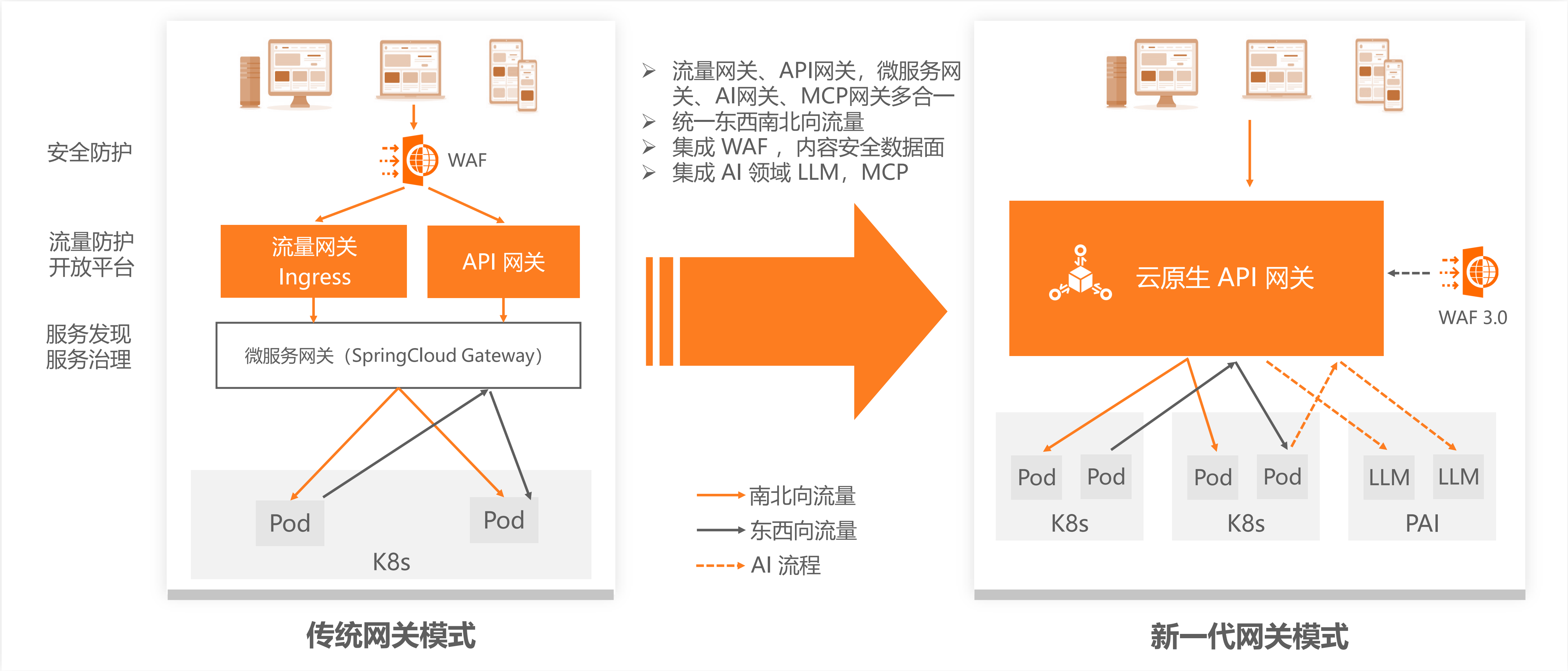


Sim Sandbox on FC 优势

- 函数计算FC需要支持Isaac Sim/IsaaC Lab环境，既可以一键拉起Isaac Sim应用（类似一键拉起ComfyUI应用一个逻辑），会落在FunctionAI应用中。
- 使用函数计算FC异步任务逻辑，天然具备任务管理能力，客户根据持有的卡数发任务即可，没卡的任务在队列里排队等待。任务的状态，回调机制都使用函数计算FC自身的能力。
- 函数计算FC除了支持CPU以外，也支持GPU，且适配了大多数常用卡型，如T4, A10, 4090 24GB, 4090 48GB, L20, H20, PPU。
- 函数计算FC异步任务和CloudFlow结合，将客户复杂的分发流程使用CloudFlow做构建。

06 云原生API网关&AI网关介绍

云原生API网关简介

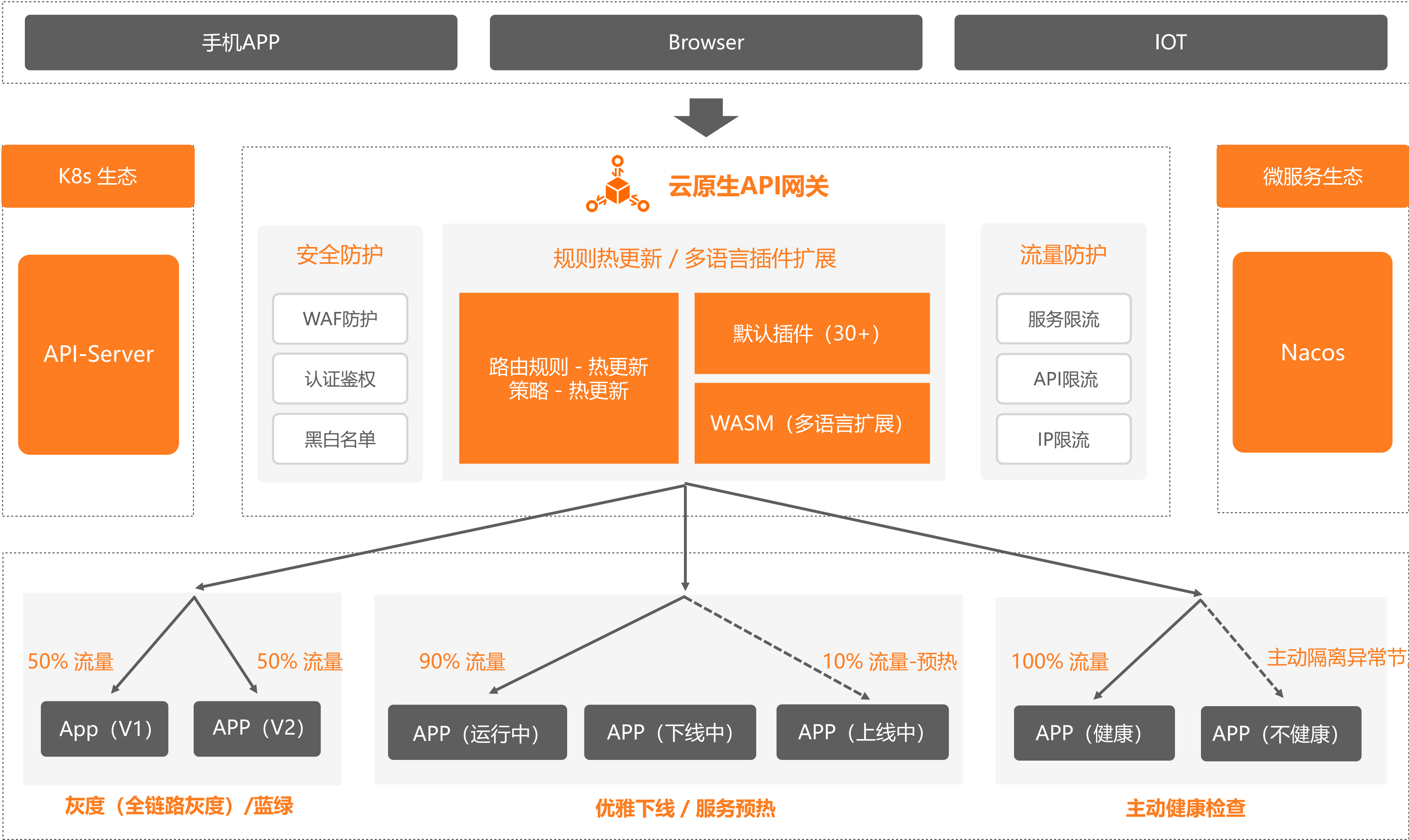


差异化竞争力：服务治理、API管理、LLM管理、MCP管理 + 基本竞争力：高性能、高可用、零信任、易扩展

云原生API网关在应用架构的核心作用 – 链接生态



云原生API网关 – 流量网关



服务发现
➢ 支持 K8s / Nacos 等主流服务发现 ➢ 深度集成函数计算FC ➢ 兼容 DNS / ECS 老的模式
服务清洗
➢ 安全防护 ➢ 流量防护
服务热更新
➢ 路由/策略更热更新 ➢ 证书热更新 ➢ 插件热更新
服务灰度
➢ 支持灰度，且支持全链路灰度 ➢ 支持蓝绿 ➢ 支持灰度观测能力
服务优雅上下线
➢ 服务下线前提前隔离流量，再停应用 ➢ 服务上线打10%流量预热
服务健康检查
➢ 隔离异常节点

云原生API网关 – API 网关



API First（前后端分离并发开发） / API 防护（默认安全/高可用） / API 货币化（扩大生态做营收）

API 货币化（开放平台）

- APP管理
- 权限管理
- 额度管理
- API 计量
- API计费

API 防护（策略管理）

- 安全防护
- 流控
- 跨域
- 超时重试
- 重写

API First（并发提效）

- API 设计
- API 文档
- API Mock
- 端代码生成
- API 测试

核心优势

智能化

- AI 辅助API设计
- AI Mock 数据
- AI 生成端代码
- AI 测试 / 诊断

策略丰富

- 内置 10+ 系统策略
- 支持 30+ 插件策略
- 支持 自定义策略（多语言）

开源开放

- 支持 Swagger（ OAS 标准）
- 支持 Ingress / Gateway API
- 开源 Higress 无厂商锁定

AI 网关



AI网关典型场景&客户

功能域	详细功能	业务场景	客户
消费者精细化管理	➤ 消费者认证 ➤ 消费者限流 ➤ 基于请求标识切换模型	精确管控企业内部、外部用户使用LLM的方式，基于请求方的特征决定使用哪个LLM，可以使用多少Token。	汽车  宝马  小鹏  深蓝汽车  蔚来  运满满
多模型统一管理	➤ 多模型厂商统一代理 ➤ 多API Key管理 ➤ 多模型切换（按模型名称，按流量）	统一管理多个LLM服务商，也包括自建LLM服务，通过多API Key管理扩展TPM限制，可以基于业务场景通过模型名称快速切换模型，比如按用户级别，按业务领域等。	
模型高可用	➤ LLM服务Fallback ➤ LLM服务健康检查 ➤ Token级别限流	增加LLM服务的健壮性，当主LLM服务不可用时告警通知并自动Fallback到备用LLM服务，通过Token级别的限流保护下游LLM服务的稳定性。	SaaS  聚水潭  畅捷通  千里马网信  众阳健康
企业级安全管理	➤ 请求/响应内容检查 ➤ IP黑白名单/全局认证鉴权（支持对接自有认证服务） ➤ 限流降级熔断	对推理的输入和输出做内容安全检查，通过IP黑白名单做请求方的管控，并且可以快捷对接客户自有的认证服务快速做集成，当有非预期流量时也可以通过限流熔断机制保护下游服务。	互娱/营销  字节跳动  今日相机  微能科技  蓝色光标
灵活扩展能力 (插件机制)	➤ 联网搜索 ➤ 推理结果缓存 ➤ 更多认证、鉴权机制	通过插件机制实现客户定制化的需求，应对更多业务场景，比如通过插件机制实现的推理时联网搜索，推理结果缓存，流式输出转非流式，更多的鉴权机制等。	零售/教育  库迪咖啡  麦当劳  可口可乐  好未来  新东方
AI场景观测体系	➤ 各维度Token消耗 ➤ LLM服务各性能指标 ➤ LLM请求日志	提供了模型维度，消费者维度，LLM服务维度的Token消耗指标，可供客户做成本管理。还提供了AI场景特有的监控指标，比如流式/非流式的RT，首Token RT等。另外更详细的推理信息也都保存在日志服务中，可供查询。	AI/游戏  清博智能  地平线  鹿客科技  途游游戏  英雄互娱
MCP Server统一管理 (MCP Hub)	➤ MCP Server代理 ➤ 普通服务转MCP Server ➤ MCP Server组装 ➤ MCP Server/MCP Tool描述信息管理 ➤ MCP Server动态发现（结合Nacos）	作为MCP Hub统一管理MCP Server，可以快速将现存的传统服务0代码改造的转换为MCP Server，也可以代理基于MCP SDK开发的MCP Server，并且在网关侧可以进一步组装MCP Server。	 爱奇艺  夺畅网络  聚水潭  东方雨虹  运满满  及刻

云原生API网关 – 内部实践

秉承着自己吃自己狗粮的原则，云原生API网关在阿里集团内部已经有很多业务在深度使用，在企业级产品能力，稳定性，性能方面已经有多个大体量业务的背书。



通义App



云原生API网关

业务Server

业务Server

云原生API网关作为流量网关，白屏操作

➢ 支持长连接SSE/WebSocket，热更新对长连接流量无损

➢ 支持流式传输，满足AI大带宽/高延时特性诉求

➢ 支持多种安全认证与限流防护

AI 应用



阿里云百炼



云原生API网关

业务网关

业务网关

云原生API网关作为流量网关，白屏操作

➢ 支持长连接SSE/WebSocket，热更新对长连接流量无损

➢ 支持流式传输，满足AI大带宽/高延时特性诉求

➢ 高可用，99.999% SLA

AI 模型服务平台



PAI



云原生API网关

Model

Model

云原生API网关作为AI网关，通过Ingress集成PAI的管控

➢ 支持1W+ 超大路由/域名规模场景，多租共享集群模式，切换到Higress后路由配置生效RT从原10分钟降到30秒内

➢ 构建完善可观测体系

AI 模型



无影

AI网关



钉钉

流量/AI网关



优酷

流量网关



通义灵码

MCP网关

07 AI网关代理LLM最佳实践

1

成本平衡问题

部署DeepSeek R1 671B满血版模型，至少需要2台8卡H20机器，列表价年度超过100W，但2台的TPS有限，无法满足生产部署中多个用户的并发请求，需要有方案找到TPS和成本之间的平衡点

2

模型幻觉问题

即使是671B的DS R1，如果没有联网搜索，依然有很严重的幻觉问题。

3

多模型切换问题

单一模型服务有较大的风险和局限性，比如稳定性风险，比如无法根据业务（消费者）选择最优模型。目前也没有开源组件和框架解决这类问题。

4

安全合规问题

企业客户需要对问答过程做审计，确保合规，减少使用风险。

5

模型服务高可用问题

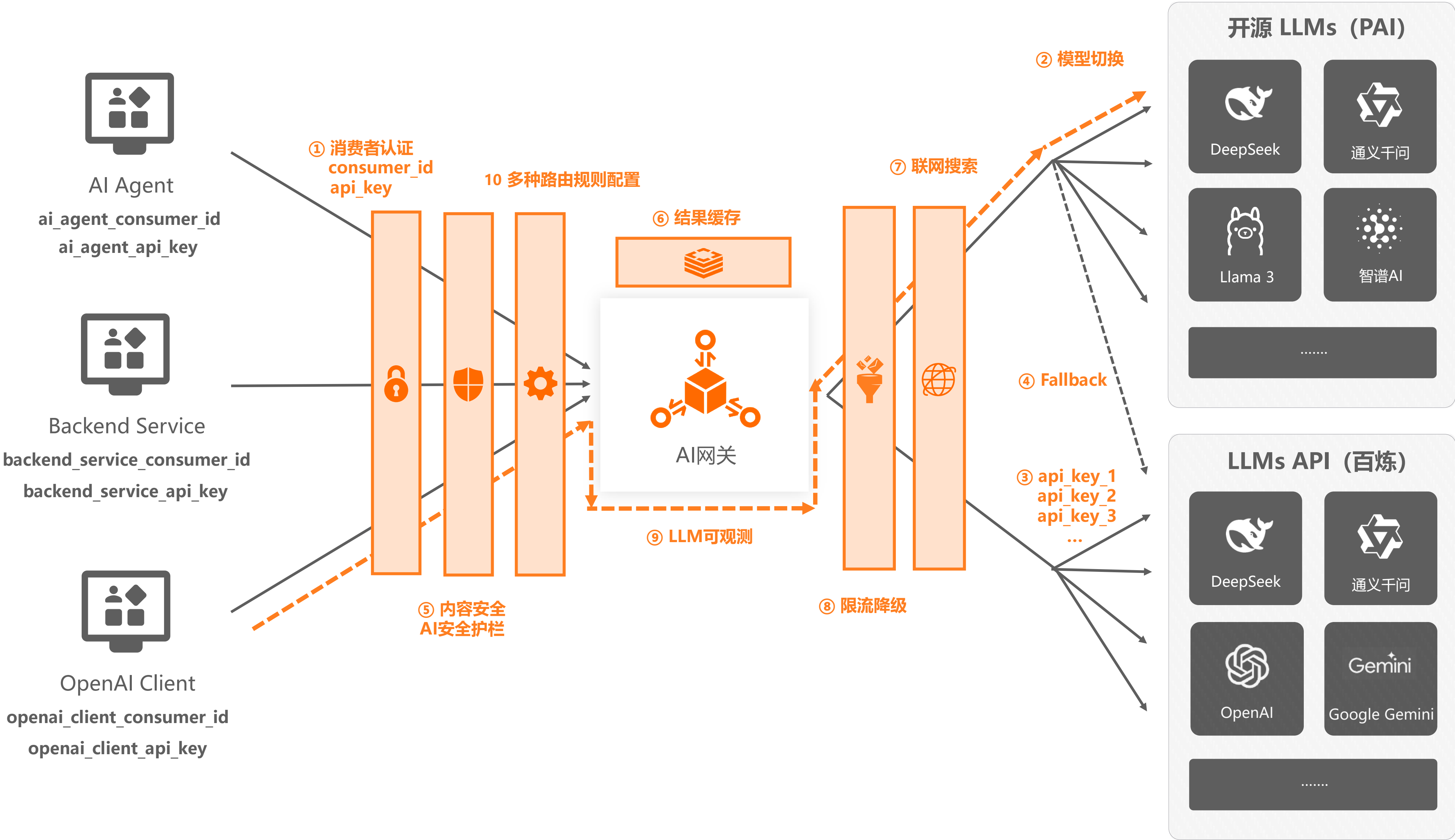
自建平台性能达到瓶颈时需要有一个大模型兜底方案，提升客户大模型使用体验。

6

闭源模型QPS/Token限制问题

商业大模型都有基于API Key维度的QPS/Token配额限制，需要一个好的方式能够做到快速扩展配额限制。

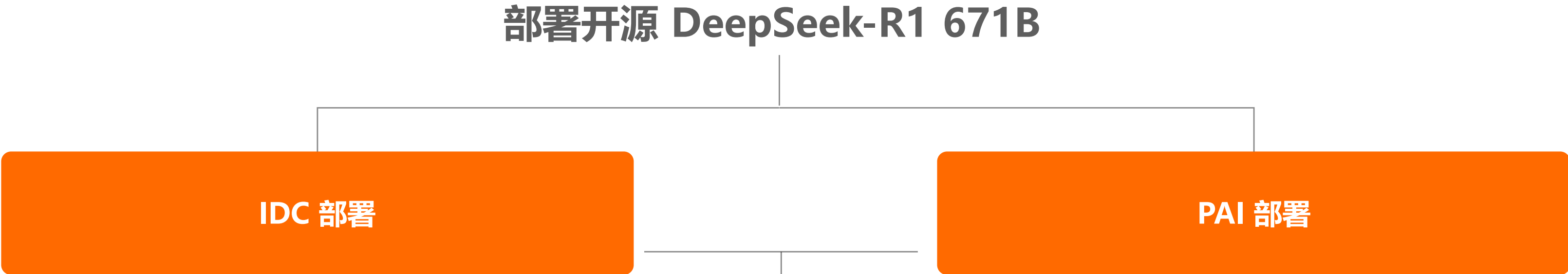
AI网关代理LLMs方案



云原生AI网关代理LLMs方案的核心收益



趋势



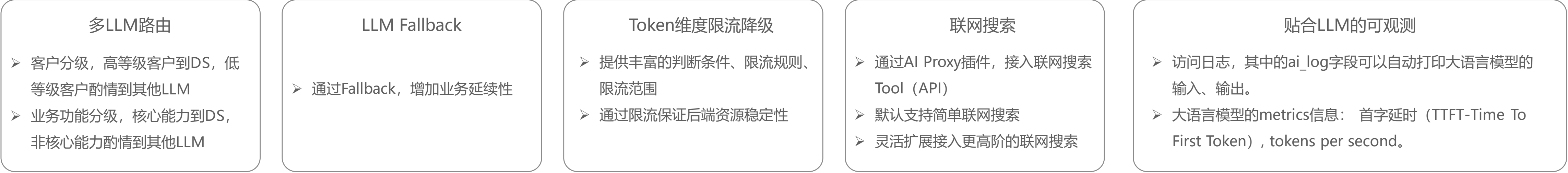
成本&性能

至少 2 台 8 卡 H20
15 TPS 15 Output Token / S
100w+ / 年

面临的问题



收益



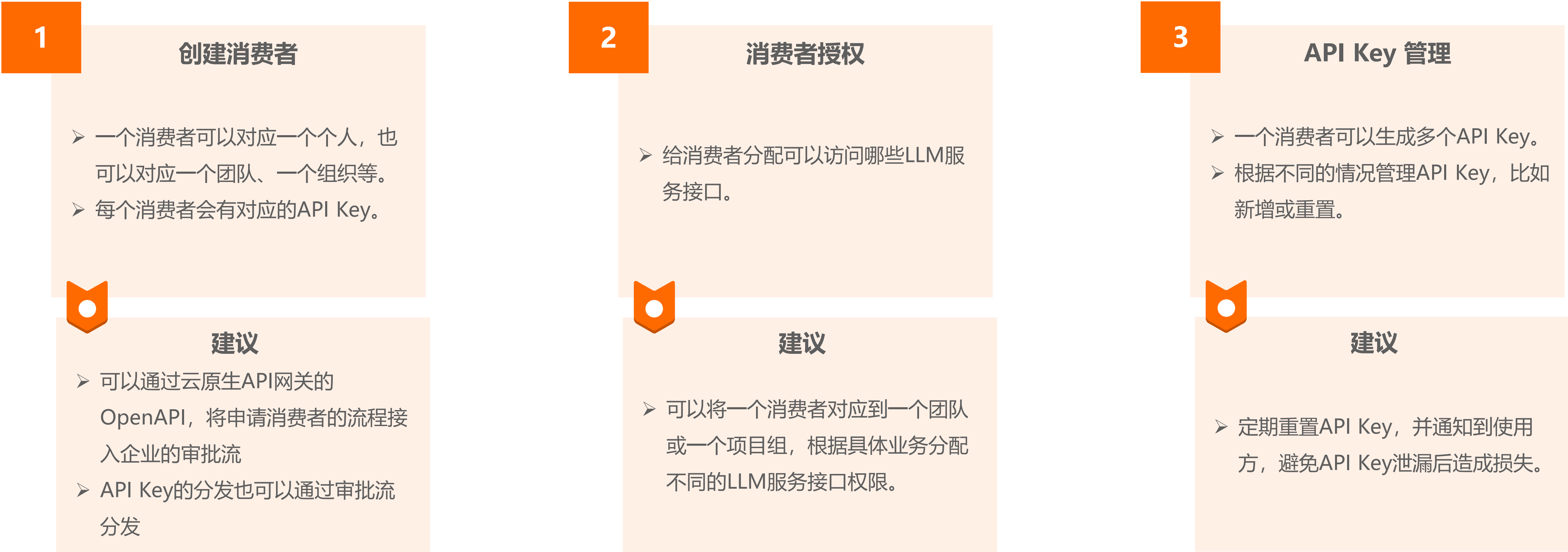
解决用户管理失控问题

核心问题1：我以什么样的方式将LLM服务和能力暴露给大家呢？

解法：OpenAI API的协议基本已经是标准协议，目前市场面上几乎所有的LLM都支持OpenAI API协议。所以提供遵循OpenAI API协议的HTTP接口就可以让企业员工通过各种方式使用LLM服务和能力。

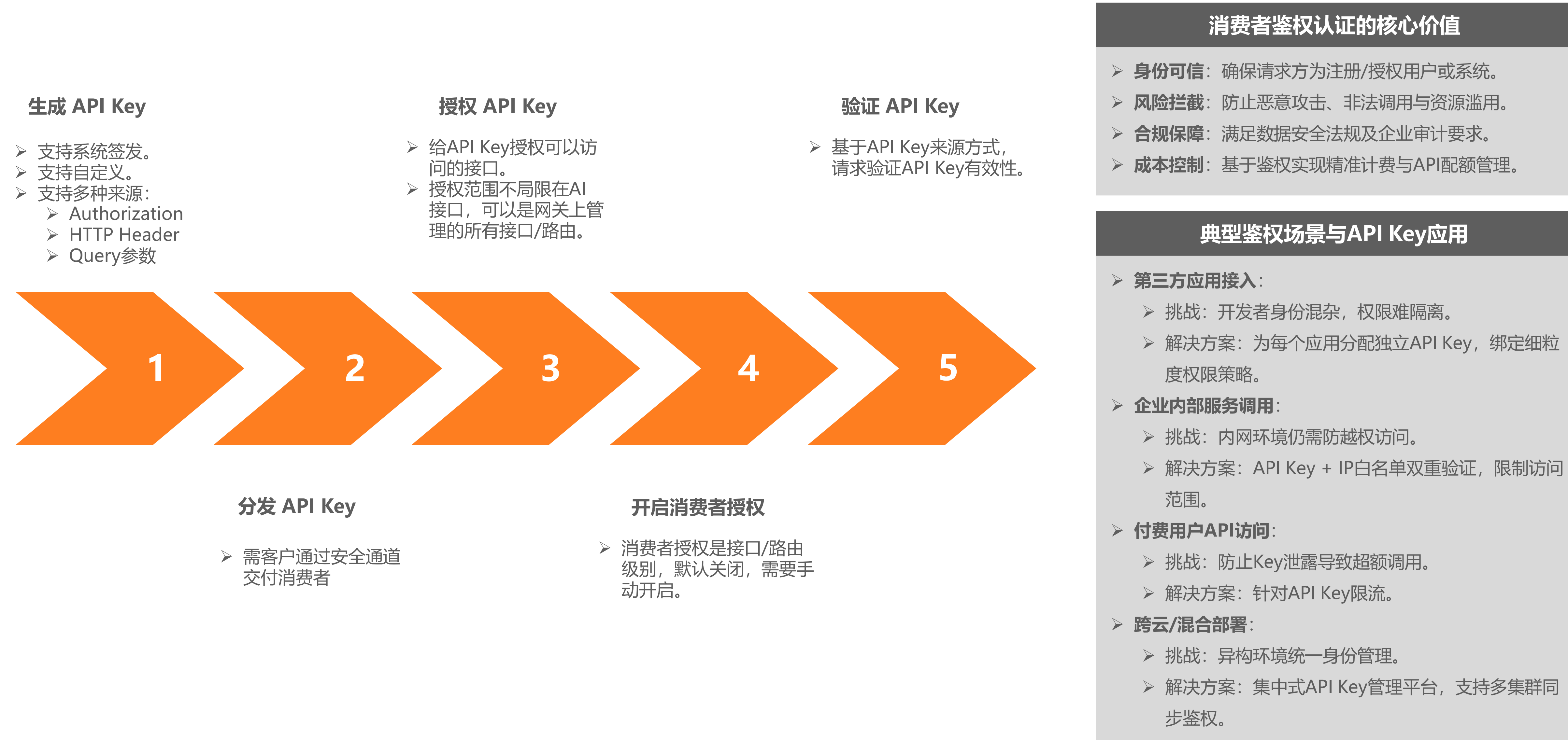
核心问题2：企业内部部署DeepSeek R1 满血版，公司好几千人，但GPU资源有限，如何限制用户？

解法：AI 接口一旦暴露出去，基本上不可能只让一小部分人知道，所以需要对访问LLM服务的用户做以限制，只让能访问的人访问，不能访问的人即便知道了接口也无法访问。



消费者鉴权认证

云原生API网关支持全局认证、路由配置认证和**消费者鉴权**，实现对API访问的控制、安全性和策略管理，确保只有授权的请求才能访问服务。



解决同一域名访问不同模型的问题

核心问题1： 公司GPU资源有限，部署了满血版DeepSeek R1，还有其他一些小模型以及使用百炼的模型服务，现在域名都不统一，分发、管理、集成的成本都很高，如何使用同一个域名来访问不同的模型？

解法：

- 满血DS R1和其他模型或者闭源LLM API服务共存，保持同一个API接口，不同业务通过请求中的模型名称，切换不同的模型。
- 满血DS R1和其他模型或者闭源LLM API服务共存，保持同一个API接口，不同业务通过请求中（Header，Cookie等）携带的业务标识，匹配到不同的模型。

1

维护多个模型服务

- 无论是PAI上部署的，IDC部署的，还是闭源LLM API，都可以作为模型服务被维护在AI网关。

2

AI API代理多个模型服务

- 使用多模型服务类型创建AI API，在一个AI API中可以添加多个模型服务。
- 模型名称通过Glob语法进行匹配。

3

同一个API请求不同模型

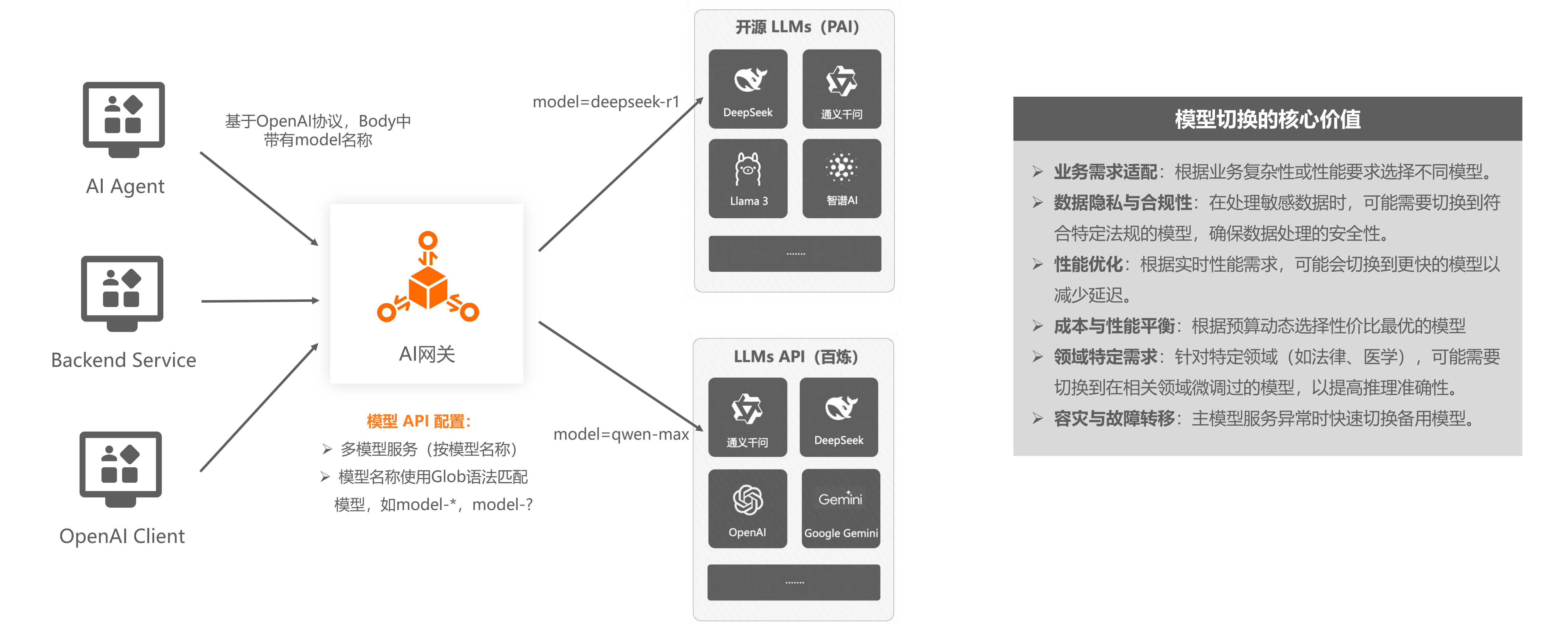
- 同一个API，不同业务传入不同的model name，即可实现模型切换。

建议

- 优先推荐使用模型名称匹配切换的模式，更遵循OpenAI协议。

模型切换

AI网关支持基于模型名称做不同后端模型的切换，实现同一个接口对接多种LLM服务（百炼，PAI，IDC）。



解决LLM托管平台/闭源LLM QPM/Token限制的问题

核心问题： 我们使用LLM托管平台上提供的DS R1 671B 模型的API，但是有QPM和TPM的配额限制，不能满足业务需求，但是每次升配很麻烦。

解法：

- 目前所有的模型托管平台都有QPM和TPM的限制，并且有些平台是很难升配这个限制的，所以大多数用户都会选择申请多个帐号（API Key），变相的撑大这个配额限制，但缺点是在业务里管理多个API Key是一件很麻烦的事。
- 对输入/输出内容做缓存，减少对模型服务的请求次数以及Token消耗，从而提升业务侧的请求性能。

1

模型服务支持多 API Key

- AI网关，每个模型服务都可以配置多个API Key。
- 每次请求会轮询拿API Key，对模型服务做请求。

2

API Key可实时维护

- 当监控到API Key配额水位较高时，可以实时动态添加模型服务的API Key。



建议

- 通过AI网关OpenAPI将添加API Key的行为集成到客户自己的自动化平台中。

3

AI API维度结果缓存

- AI API维度支持将输入和输出缓存到Redis，只需要配置Redis地址即可
- 支持精确匹配
- 支持向量化检索匹配

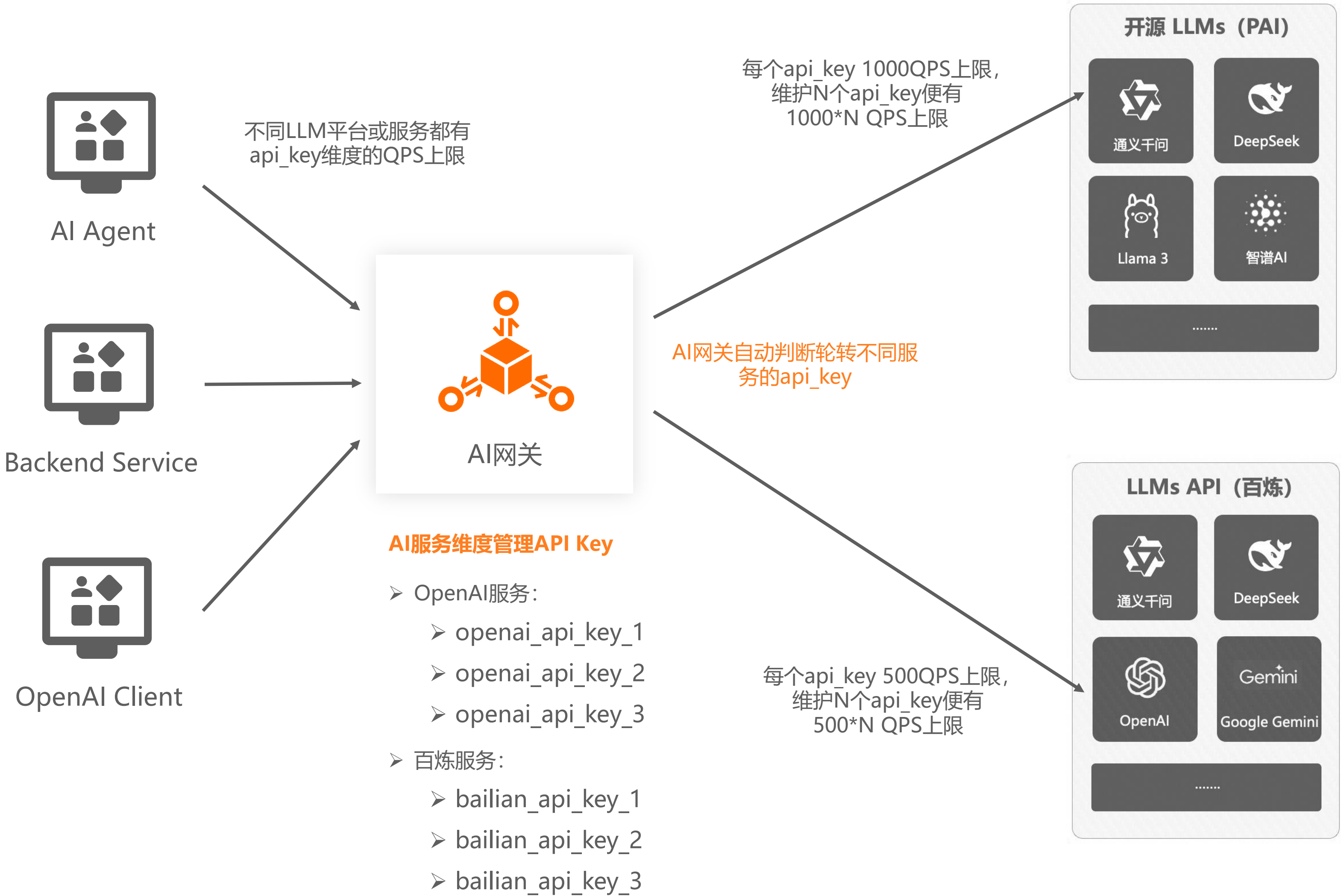


建议

- 在非常垂直类的应用场景下适合开启结果缓存，但建议开向量化检索匹配
- 在非常垂直类，问题和答案非常固定的应用场景下可以开精确匹配
- 在泛业务场景下开启结果缓存可能会降低推理精度或准确性，需要结合业务判断和考量

多API Key 管理

AI网关支持管理多个不同LLM托管平台，闭源LLM的API Key，突破LLM托管平台，闭源LLM的QPS限制。



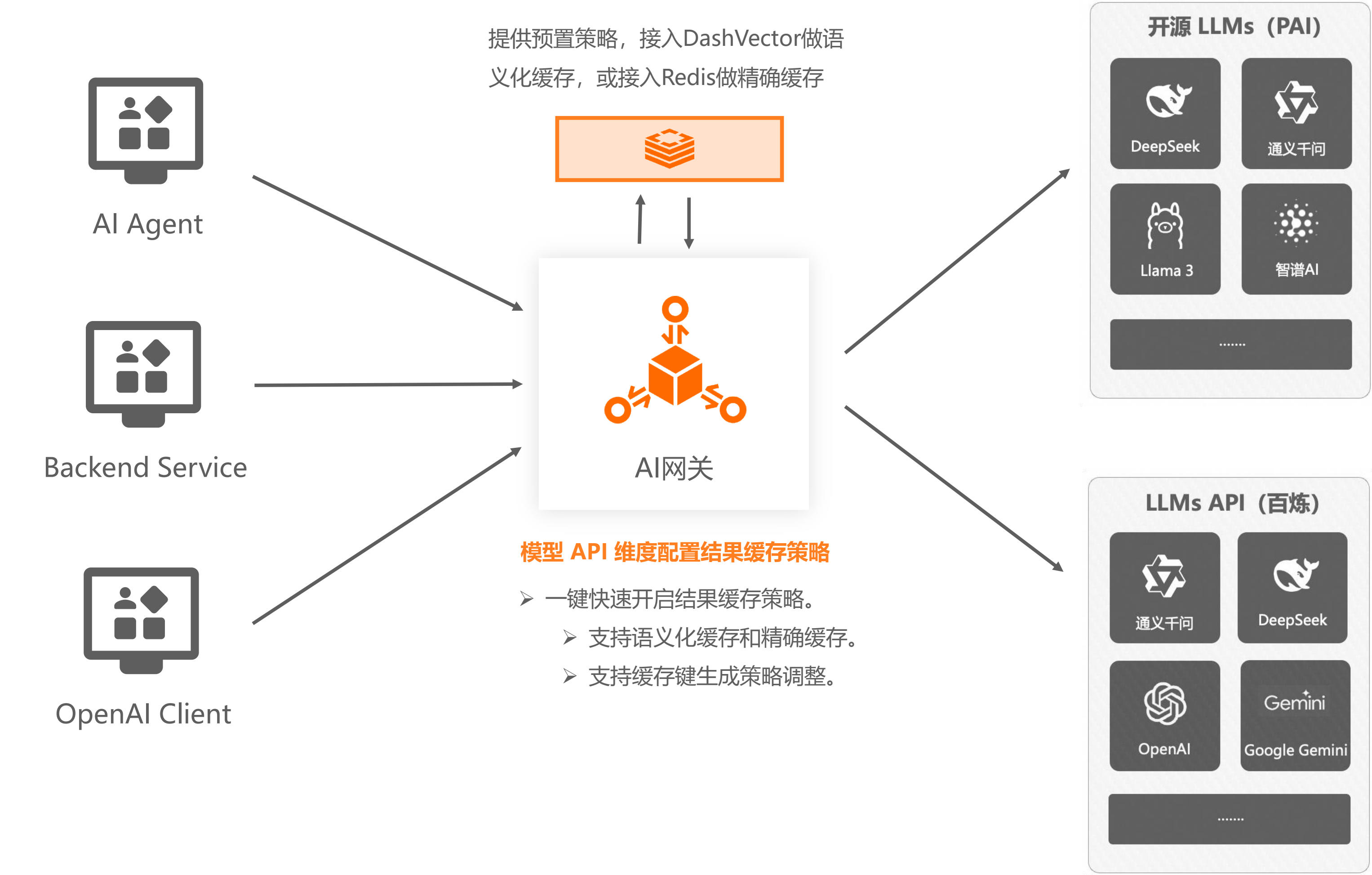
多API Key管理的核心价值

像ChatGPT，豆包这类闭源LLM，或者百炼这种托管LLM平台，都是以提供API的方式供大家使用LLM的能力，但是受限底层GPU资源的压力，以及整体平台的稳定性，每个用户都有请求QPS的最大限制（基于平台的API Key的维度），且上调比较困难。

- **突破QPS上限：**通过管理闭源LLM或LLM托管平台的多个API Key，变相提升QPS上限，提升业务性能。

结果缓存

AI网关提供了扩展点，可以将请求和响应的内容缓存到Redis，提升推理效率。



结果缓存的核心价值

- **提高效率**：如果相同的输入反复出现，缓存可以避免重复运行模型，从而加快响应速度，特别是在处理常见问题时。
- **降低成本**：减少模型调用次数可以节省计算资源，尤其对大型模型来说成本较高。
- **保持一致性**：缓存确保相同输入产生相同输出，有助于测试和合规性场景。

一键开启结果缓存

缓存方式

☒ 语义化缓存 ☐ 精确缓存

缓存键生成策略

☒ 只取最新提问 ☐ 整合历史提问

文本向量化

AI 服务

请选择

模型名称

请选择模型名称

超时时间

5000

ms

向量数据库

服务提供者类型

向量检索服务 dashvector

前往控制台

服务地址

请输入服务地址

Collection 名称

请输入 Collection 名称

API Key

请输入 API Key

显示

向量相似度阈值

0.8

超时时间

3000

建议范围: 0.8 ~ 0.9;

解决模型服务高可用的问题

核心问题： 我们公司的主力模型是PAI上部署的DS R1 671B，但GPU资源并不是基于流量峰值储备的，所以当高峰期时，DS服务会请求失败，有什么办法可以保证业务健壮性？

解法： 有两种做法，并且可以搭配使用：

- 可以构建多个个兜底模型服务，如果要想保证模型一致，可以主力使用PAI上部署的，兜底使用百炼平台提供的。实现当PAI上部署的DS服务请求失败时，Fallback到百炼平台托管的DS R1 服务。从而保证业务的连续性和健壮性。
- 通过基于Tokens的限流策略，解决Burst流量，保护后端模型服务。

1

维护多个模型服务

- 无论是PAI上部署的，IDC部署的，还是百炼LLM API服务，都可以作为模型服务被维护在AI网关。

2

开启AI API限流策略

- AI API限流策略需要配合Redis实现，但是只需要开通Redis和在AI网关侧配置即可。
- 支持多种限流判断条件：Header，Query参数，Cookie，消费者，客户端IP

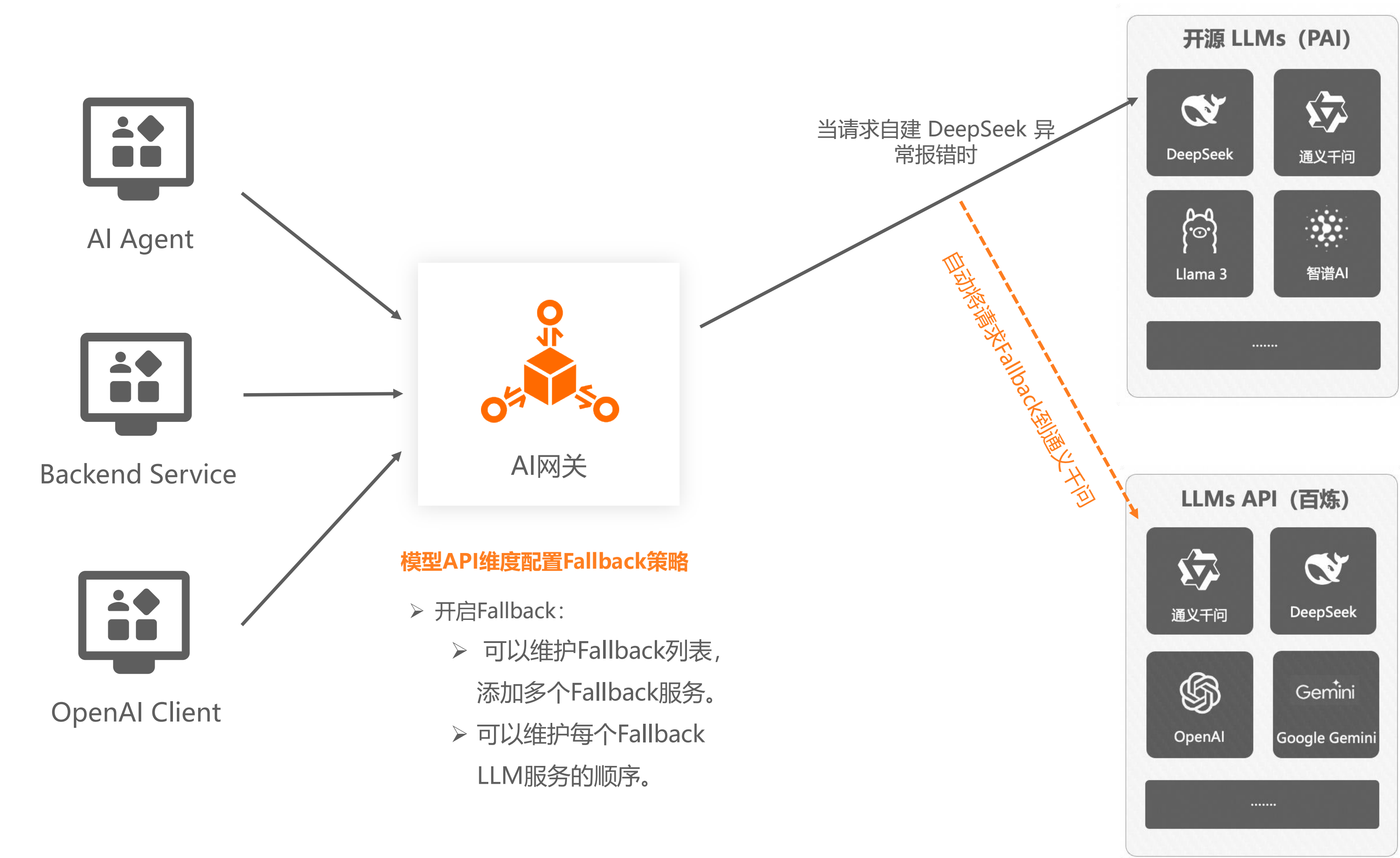
3

开启AI API Fallback策略

- AI API一键开启Fallback策略。
- 当主LLM服务出现异常后Fallback到指定的其他LLM服务。
- 支持配置多个Fallback模型服务。

LLM服务Fallback

AI网关支持当某LLM服务请求失败后，Fallback到指定的其他LLM服务，以保证服务的健壮性和连续性。



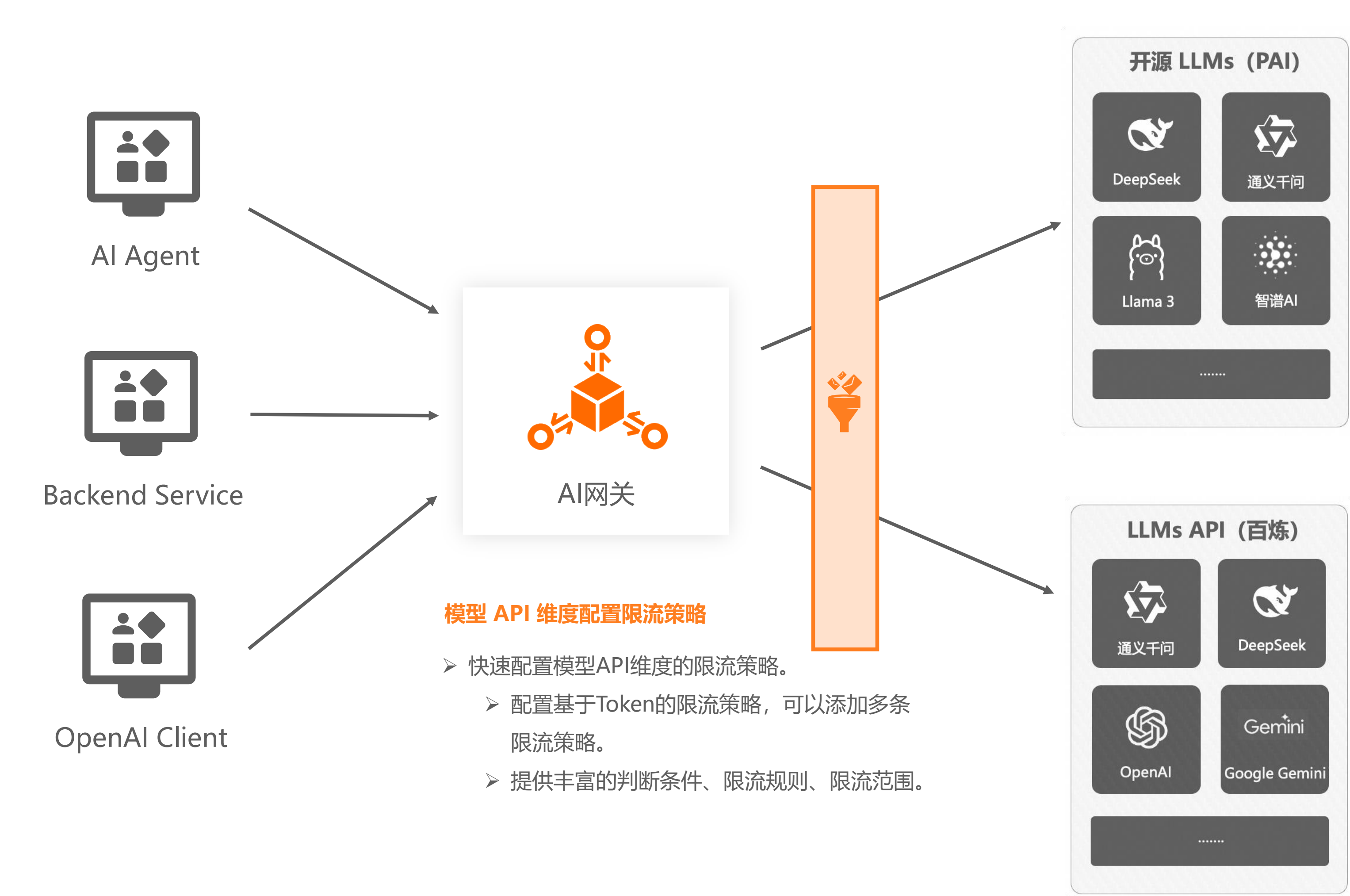
LLM服务Fallback的核心价值

当主LLM服务因为各种原因出现异常，不能提供服务时，网关侧可以快速将请求Fallback到配置的其他LLM服务，虽然可能推理质量有所下降，但是保证了业务的持续性，争取了排查主LLM服务的时间。

- **配置多个Fallback LLM服务**：通过管理闭源LLM或LLM托管平台的多个API Key，变相提升QPS上限，提升业务性能。

基于Token维度的限流降级

除了传统的QPS限流降级以外，AI网关支持更贴合LLM推理场景的Token维度的限流能力。



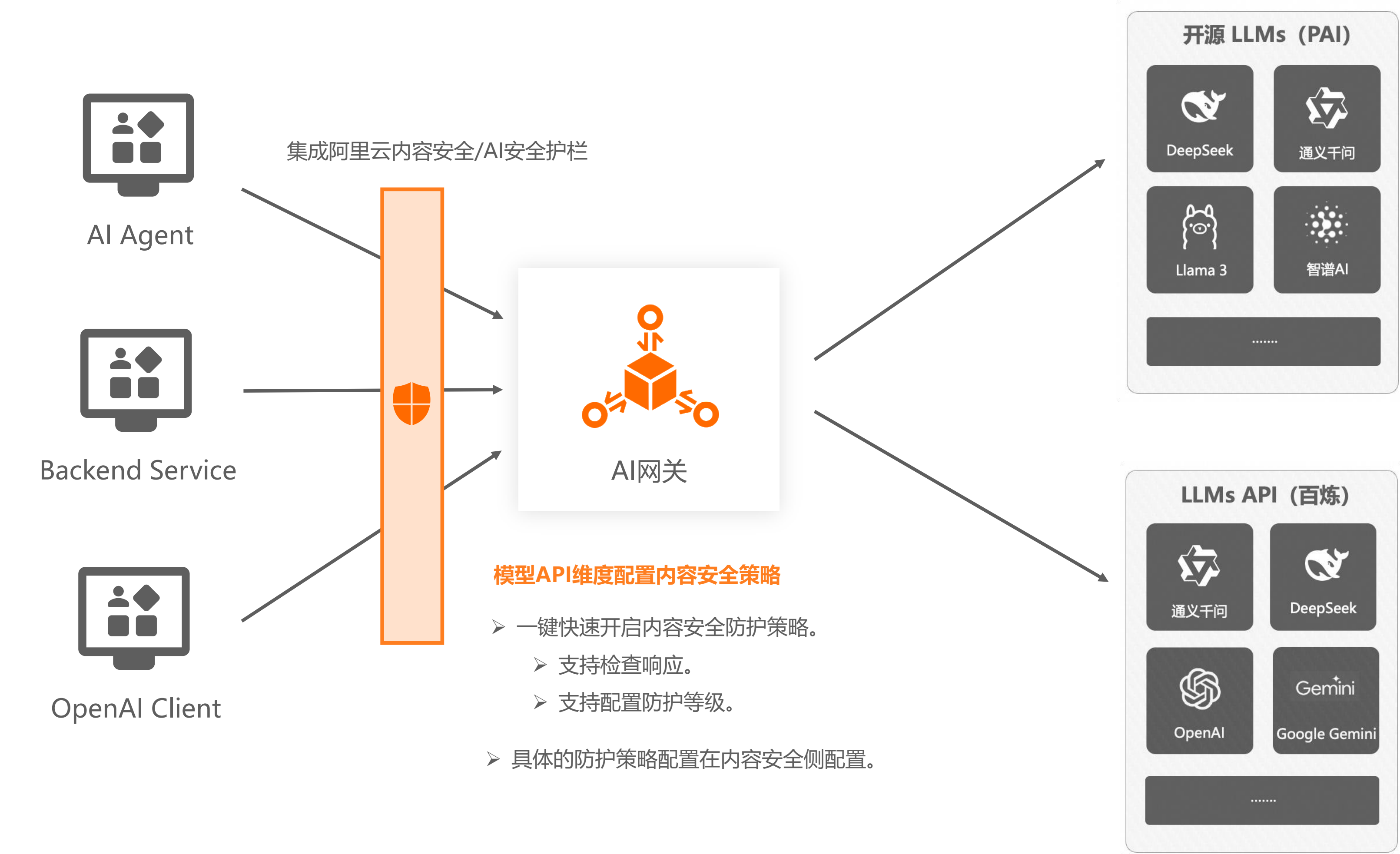
基于Token维度限流的核心价值

- 成本管理：LLM的费用通常基于Token数量计算，限流帮助用户避免超支。例如，服务提供商可能按Token使用量提供不同定价层。
- 资源管理：LLM需要大量计算资源，限流防止系统过载，确保所有用户都能获得稳定性能，尤其在高峰期。
- 用户分层：可以基于ConsumerId或者API Key进行Token限流。
- 防止恶意使用：通过限制Token数量来减少垃圾请求或攻击。

限流策略

- 判断条件：
 - 支持按请求Header判断。
 - 支持按请求Query参数判断。
 - 支持按请求Cookie判断。
 - 支持按客户端IP判断。
- 限流规则：
 - 精确匹配。
 - 前缀匹配。
 - 正则匹配。
 - 任意匹配。
- 限流范围：每秒、每分钟、每小时、每天。

AI网关和内容安全集成，在网关侧实现基于阿里云内容安全检测大模型的输入输出，保障AI应用内容合法合规。



内容安全的核心价值

- **防止攻击**：验证输入可以阻止恶意提示注入，防止模型生成有害内容。
- **维护模型完整性**：避免输入操纵模型，导致错误或偏见输出。
- **用户安全**：确保输出没有有害或误导性内容，保护用户免受不良影响。
- **内容适度**：过滤掉不适当的内容，如仇恨言论或不雅语言，特别是在公共应用中。
- **法律合规**：确保输出符合法律和伦理标准，尤其在医疗或金融领域。

一键开启内容安全防护

内容安全防护

防护服务

https://s.com

默认填充内容安全的URL，您可以[参考文档](#)

防护服务

内容安全

AI 安全护栏

检查请求

开启

关闭

检查响应

开启

关闭

检查响应应用于检查大模型的回答内容是否合规，生效时会使流式响应变为非流式

防护等级

中

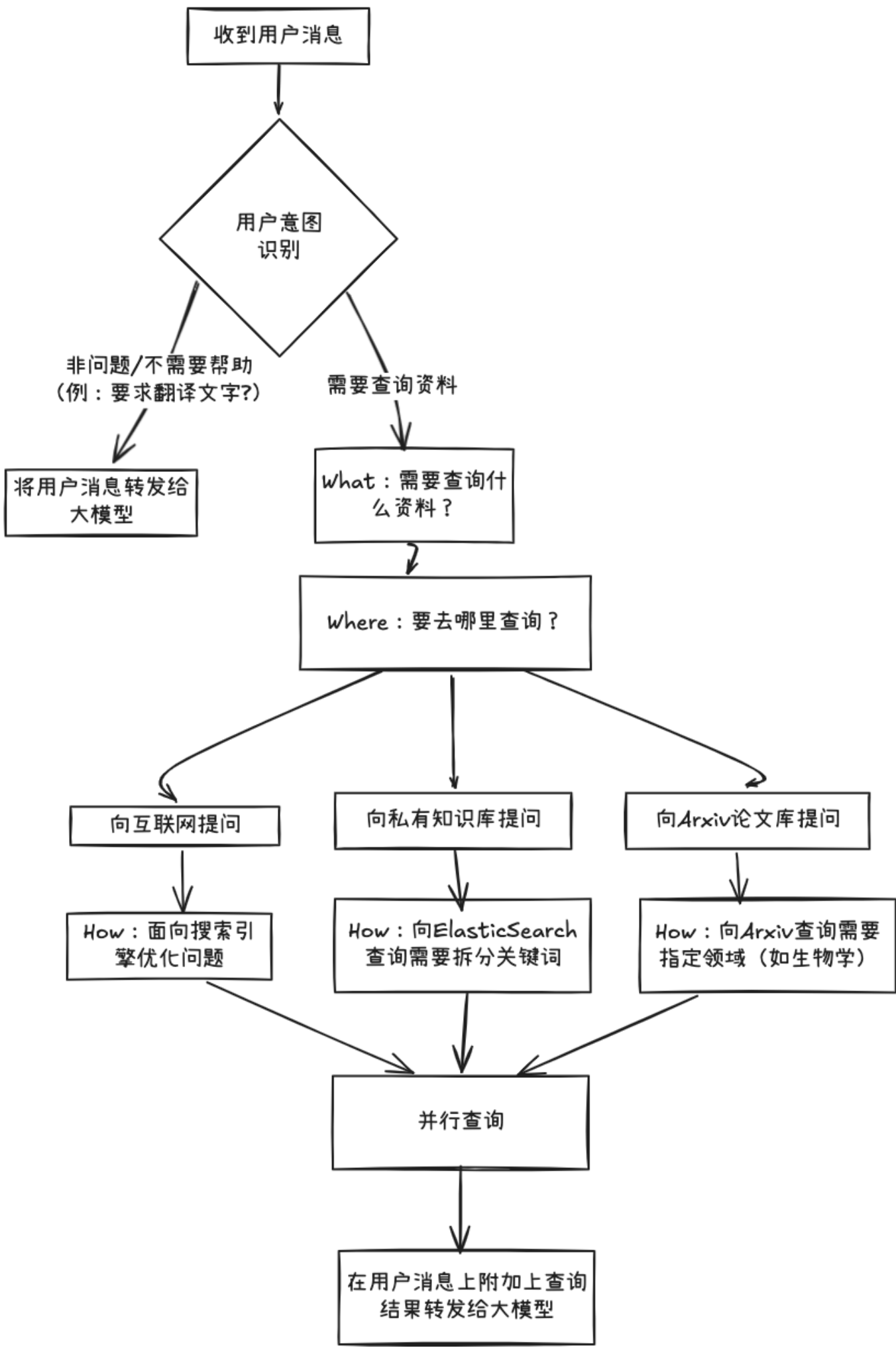
保存

取消

解决大语言模型幻觉的问题

核心问题：公司部署了DeepSeek R1 671B的模型，但推理的结果和DS官网推理的结果有差距，似乎不满血？

解法：推理的结果和DS官网推理的结果有差距大概率是因为DS官网开启了联网搜索。DeepSeek R1 671B的模型推理能力是很强，但训练的数据也是有限的，所以要解决幻觉还需是要在推理前先搜索和处理出比较确切的信息后，再由DS R1推理，所以联网搜索是非常关键的。目前模型托管平台提供的DS R1 API和自己部署的DS R1都需要自己实现联网搜索。



1

支持夸克联网搜索

- AI网关在AI API维度集成了夸克和必应的联网搜索能力
- 模型API策略中一键开启，快速配置

2

搜索结果自动融合

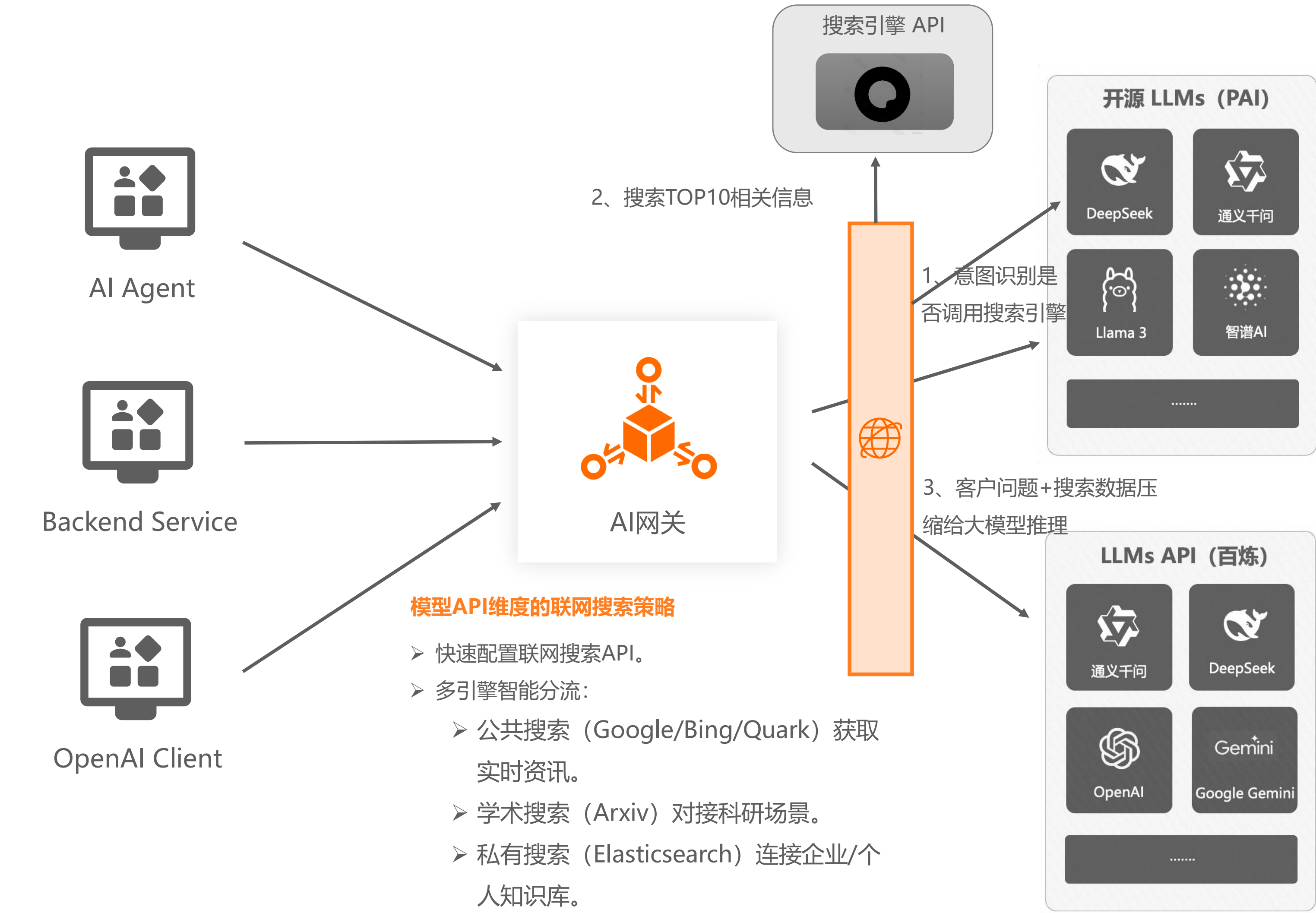
- 搜索策略有多种配置项。
- 搜索结果自动融合进输入的Prompt，无需用户额外处理。

3

问题意图识别

- 默认使用小模型对用户的问题做意图识别，避免无效的联网搜索

AI网关提供插件机制，可以快速对接联网搜索Tool（API）。大幅优化LLM的推理幻觉问题。



联网搜索的重要性

虽然DS是开源的，但是大家可能忽略了一个问题，那就是联网搜索。当不开联网搜索时，DS的推理结果会大打折扣，所以真正意义上的满血版DS R1应该是开了联网搜索的671B R1模型。而目前各个托管DS满血模型的平台都不支持联网搜索，比如百炼提供的，Ollama提供的所谓满血版DeepSeek R1。所以单纯的使用DS满血推理效果也是很一般的，有很大幻觉。即便像我们的AI Studio自己实现了联网搜索能力，效果也不及DS官网实现的。

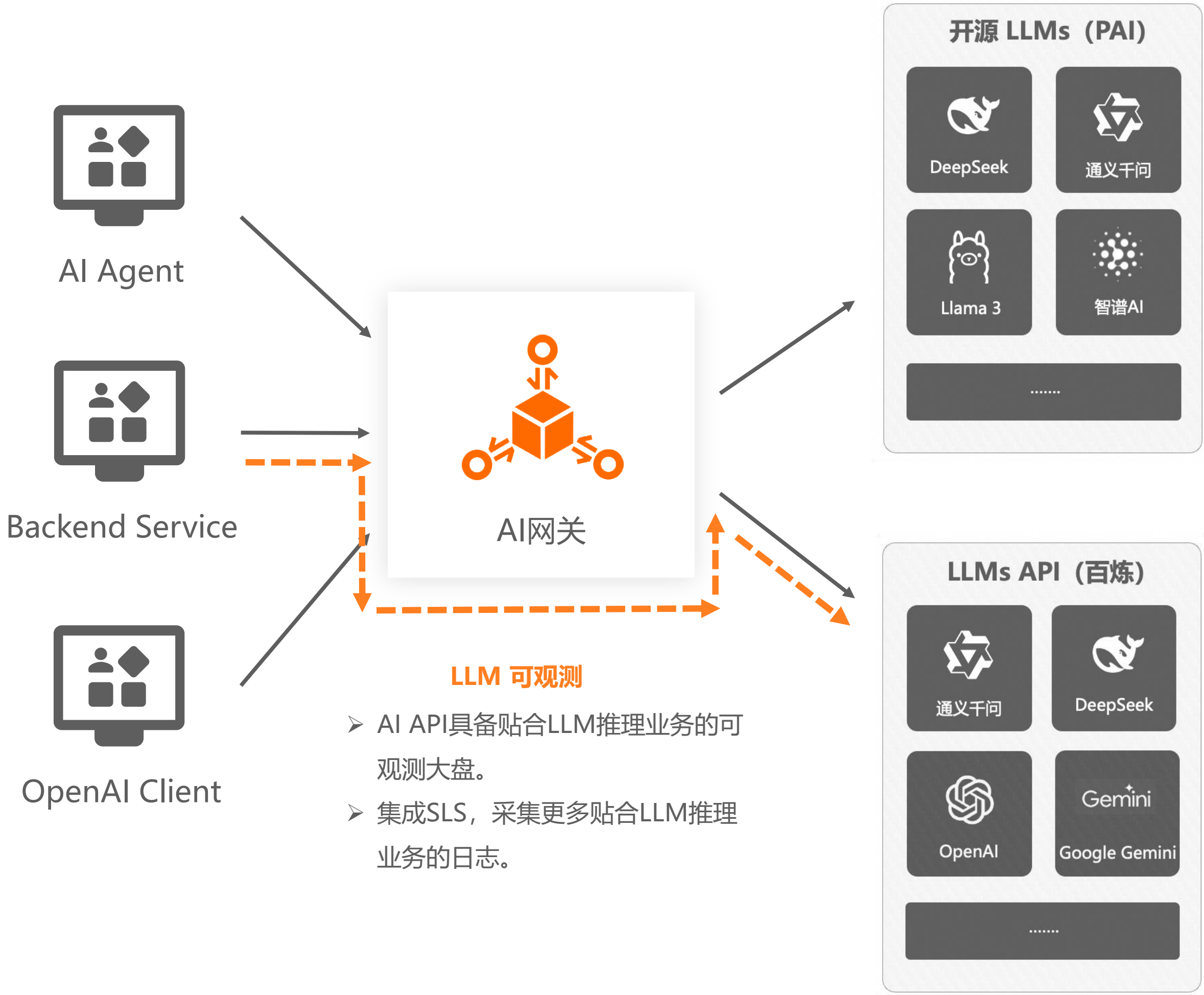
参考：<https://mp.weixin.qq.com/s/Q99LtM7wxgMCIHn6a8otg>

搜索增强核心思路

- LLM 重写 Query：基于 LLM 识别用户意图，生成搜索命令，可以大幅提升搜索增强效果。
- 关键词提炼：针对不同的引擎，需要生成不同的提示词，例如 Arxiv 里英文论文居多，关键词需要用英文。
- 领域识别：仍以 Arxiv 举例，Arxiv 划分了计算机科学/物理学/数学/生物学等等不同学科下的细分领域，指定领域进行搜索，可以提升搜索准确度。
- 长查询拆分：长查询可以拆分为多个短查询，提高搜索效率。
- 高质量数据：Google/Bing/Arxiv 搜索都只能输出文章摘要，而基于阿里云信息检索对接 Quark 搜索，可以获取全文，可以提高 LLM 生成内容的质量。

LLM 可观测

AI网关支持在应用、网关、后端LLM服务上开启OT服务来进行全链路的跟踪，通过TraceId来串联各个地方的日志、请求参数等信息。

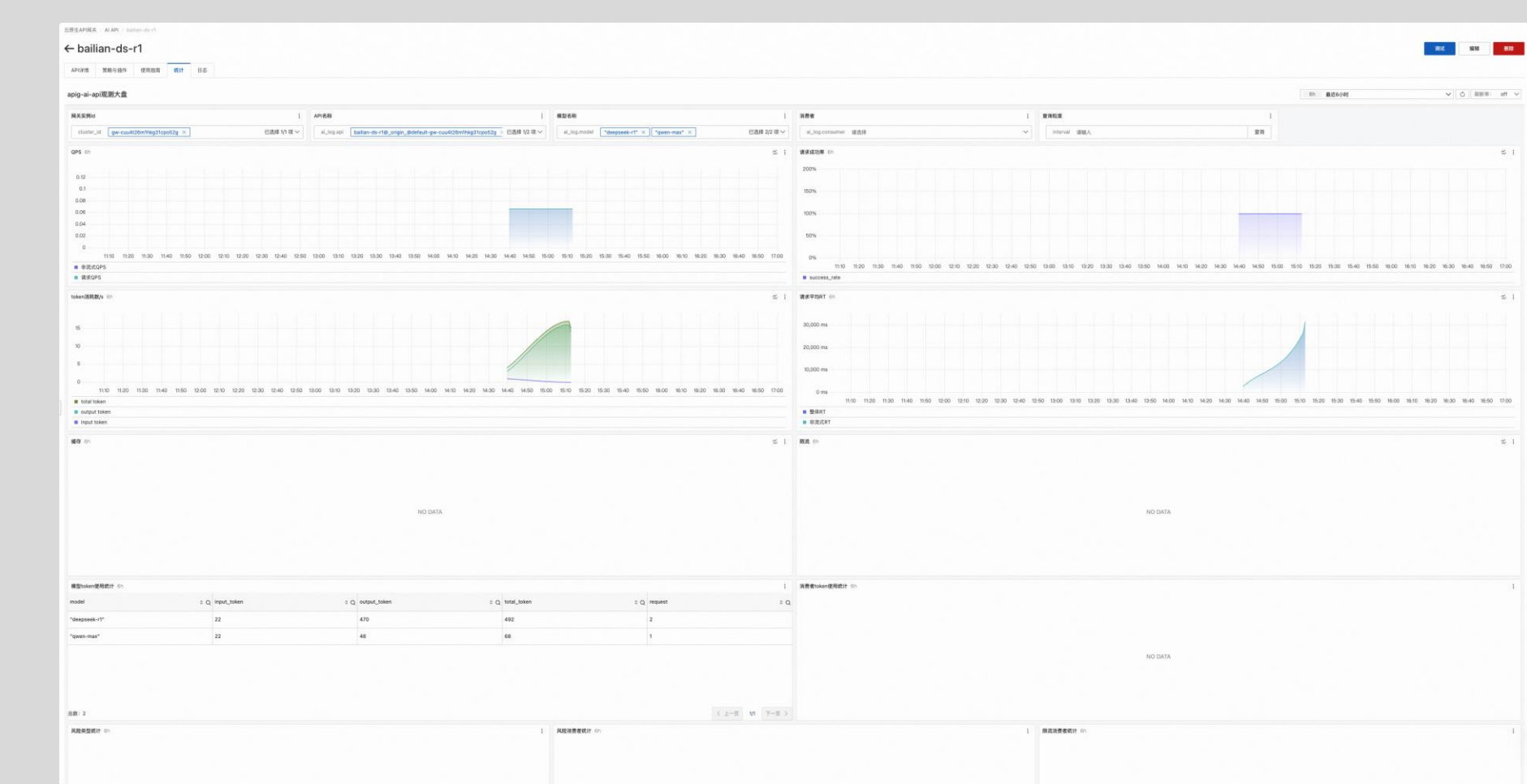


LLM推理服务日志采集

云原生API网关默认集成SLS日志服务基于日志服务提供

- 访问日志，其中的ai_log字段可以自动打印大语言模型的输入、输出。
- 大语言模型的metrics信息：首字延时（TTFT-Time To First Token），tokens per second。
- 传统指标：QPS(request per second), RT(延时)，错误率。
- 网关功能指标：
 - 基于consumer的token消耗统计(需要把consumer的header信息加到sls的日志里)
 - 基于模型的token消耗统计。
 - 限流指标: 每单位时间内有多少次请求因为限流被拦截; 限流消费者统计（是哪些消费者在被限流）。
 - 缓存命中情况。
 - 安全统计：风险类型统计、风险消费者统计。

LLM可观测大盘



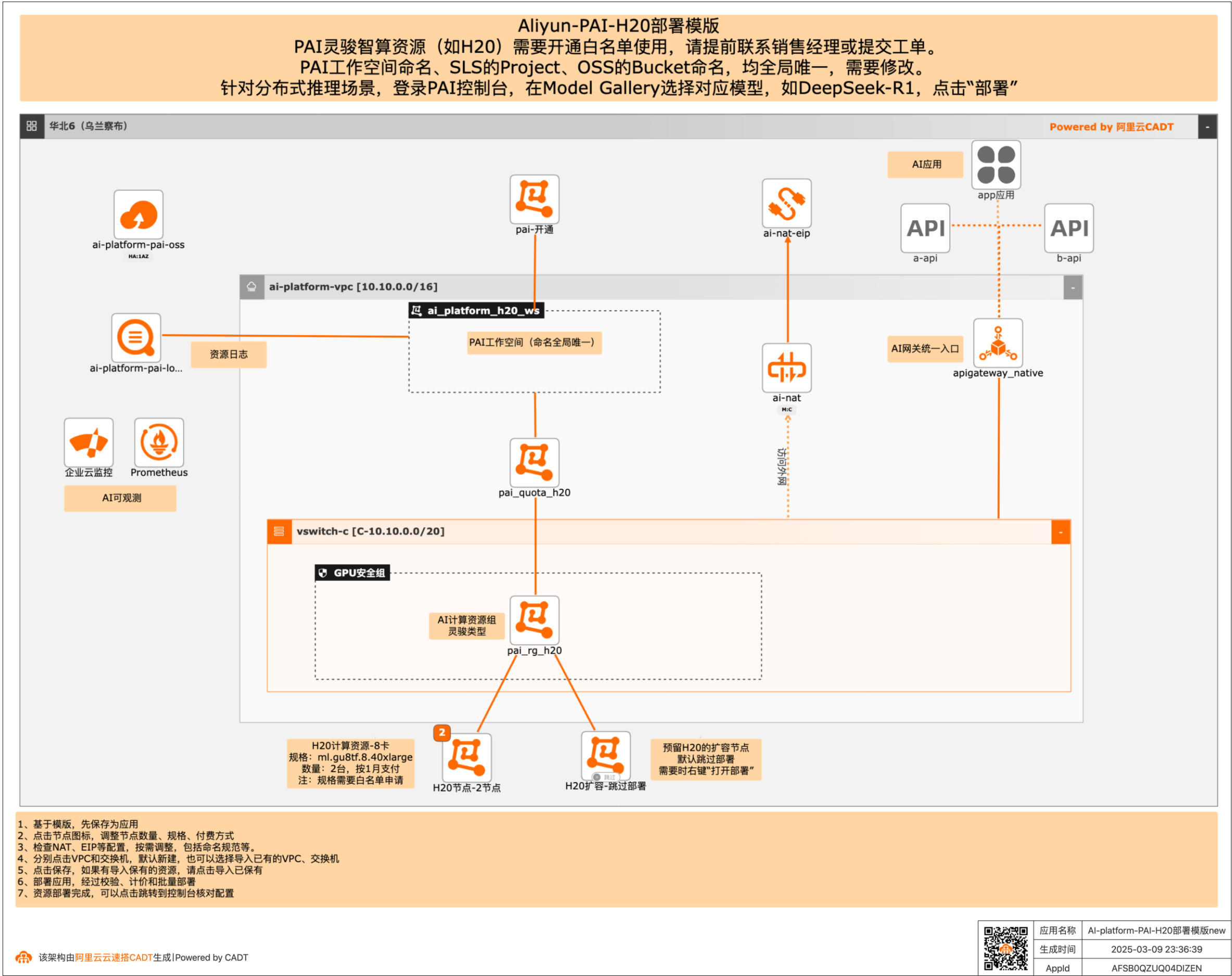
基于CADT可视化部署LLMs业务

架构要点

- 整体架构根据业务需求，部署在阿里云乌兰察布，可用区C。
- 网络规划：VPC：10.10.0.0/16，可用区C：10.10.0.0/24（可用IP数252个）
- NAT+EIP 复用现有资源，单独配置，统一给VPC内服务配置公网访问能力。
- 开通人工智能平台PAI，灵骏智算资源规划在配额（pai_quota_h20）中，并将资源配额绑定到指定的工作空间(ai_ai_h20_ws)。
- H20对应规格：ml.gu8tf.8.40xlarge，开通2台。扩容需提前报备锁定。
- 人工智能平台PAI的日志投递到日志存储SLS，包括DSW、DLC等日志。
- 开通ARMS-Prometheus，提供AI资源全链路可观测和多维度分析，开箱即用的内置大盘和告警规则。
- 基于云原生API网关提供统一网关服务，提供AI内容安全保障和模型灰度调度等。
- 架构参考CADT大模型标准模版，完成设计和参数调整，并整体校验和批量部署。

云产品列表

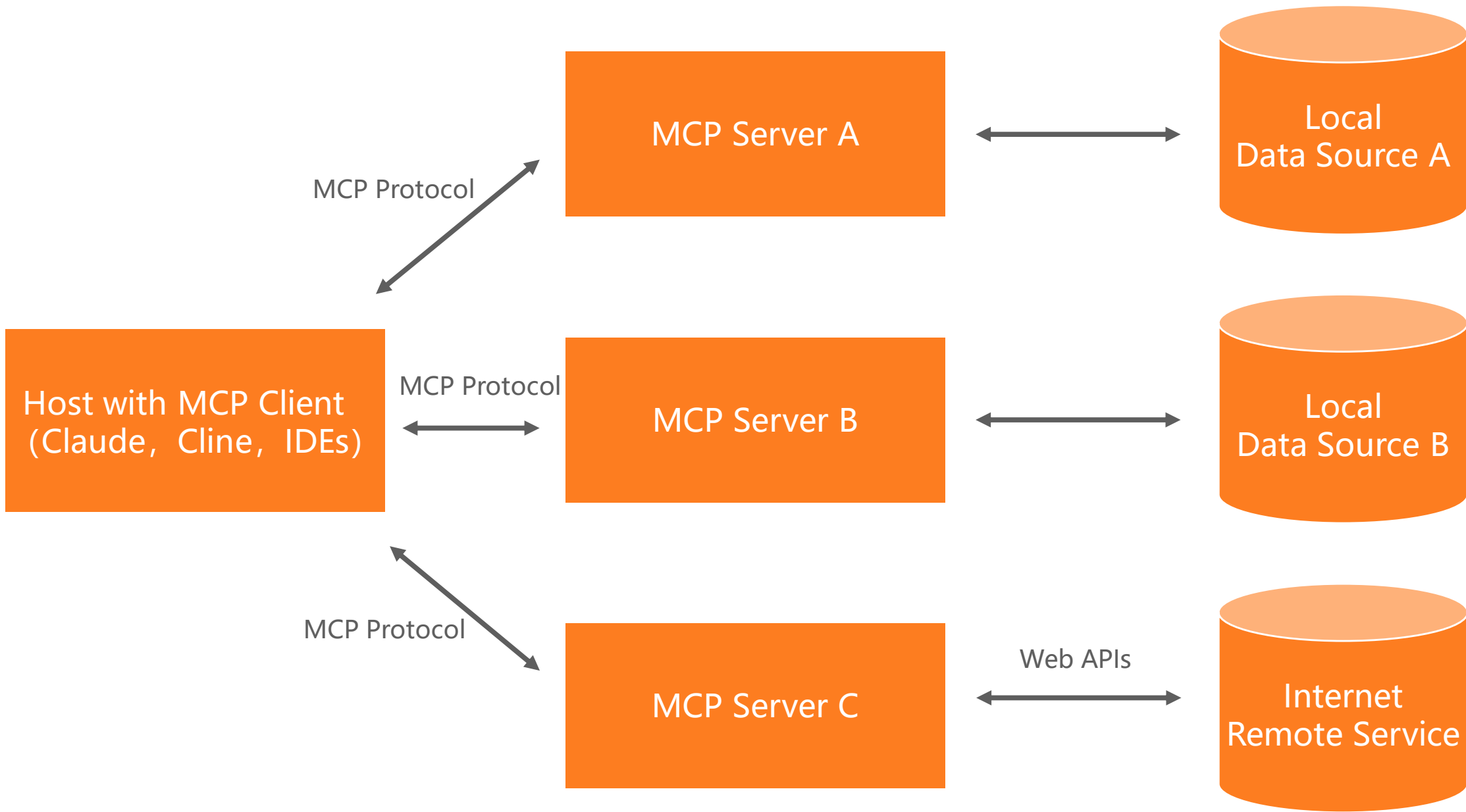
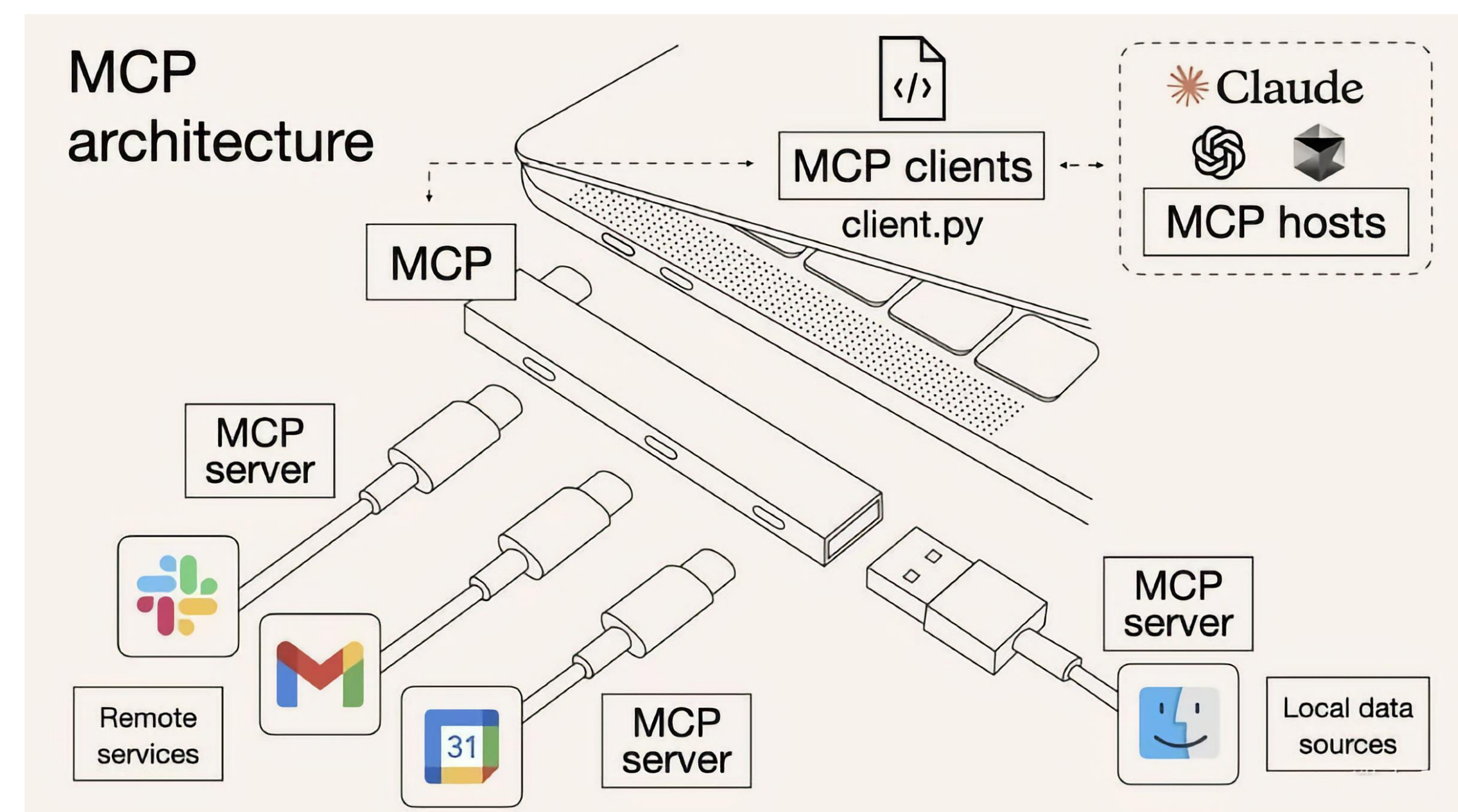
专有网络VPC，交换机、弹性公网EIP，NAT网关，机器学习PAI，PAI工作空间，资源配额，PAI资源组，GPU节点，对象存储OSS，日志SLS，灵骏安全组、Prometheus、云原生API网关、云速搭CADT。



08 MCP 概述

MCP 是什么

模型上下文协议（Model Context Protocol）是一个开源协议，由Anthropic（Claude开发公司）开发，旨在让大型语言模型（LLM）能够以标准化的方式连接到外部数据源和工具。它就像AI应用的通用接口，帮助开发者构建更灵活、更具上下文感知能力的AI应用，而无需为每个AI模型和外部系统组合进行定制集成。MCP被设计为一个通用接口，类似于USB-C端口，允许LLM应用以一致的方式连接到各种数据源和工具，如文件、数据库、API等。



标准化

MCP标准化了LLM访问外部数据的方式，简化了不同数据源和工具的集成。

模块化

MCP促进了模块化设计，允许独立开发和维护不同组件。

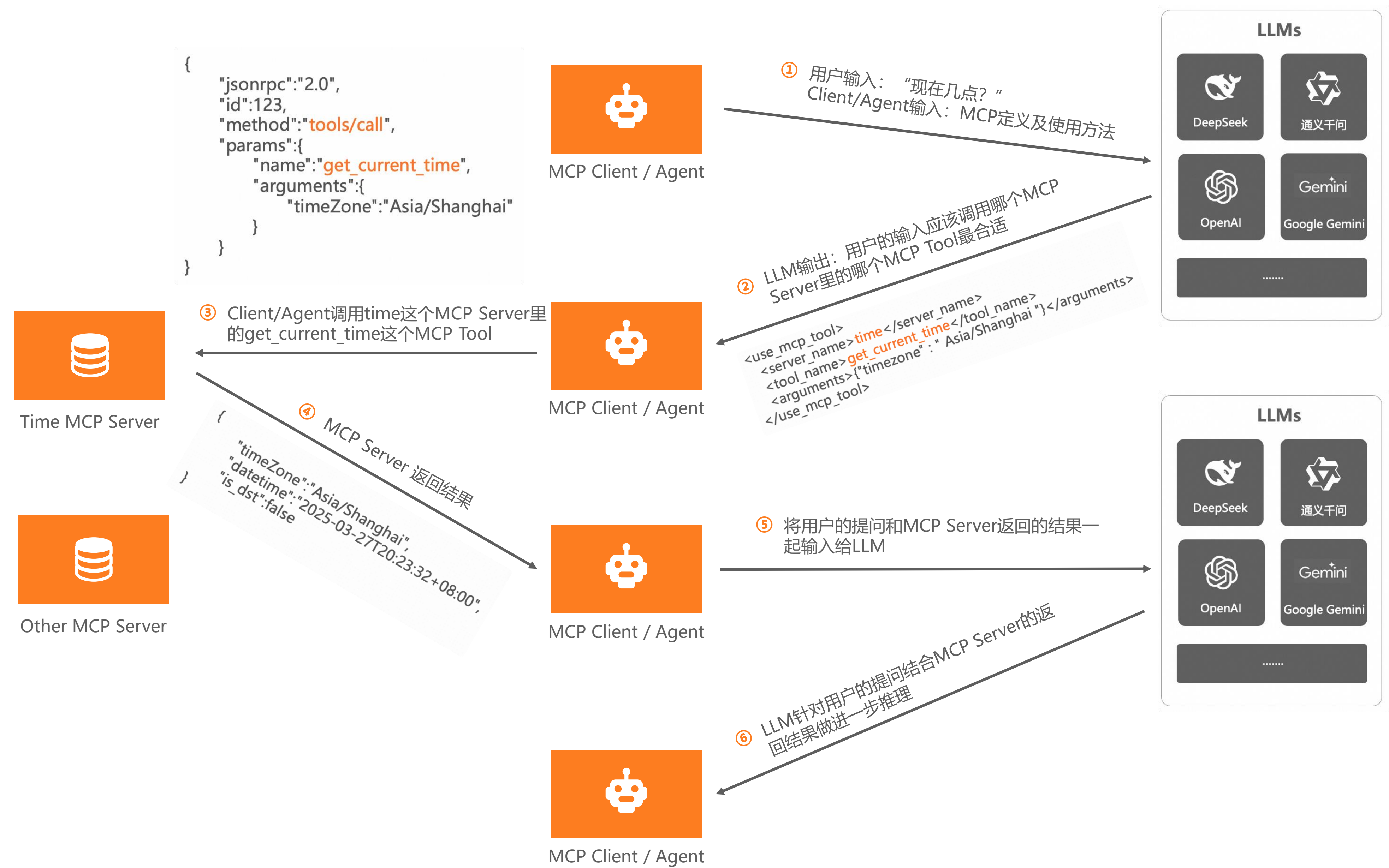
可扩展性

MCP使得添加新数据源或工具变得简单，无需大幅修改现有系统。

安全性

MCP提供结构化的访问模式，内置验证，确保数据交互安全且受控。

MCP 协议的运作机制



MCP 协议的核心



MCP不像传统的协议定义，它没有一个确定的数据结构。它的核心是**通过自然语言描述清楚有哪些MCP Server，承担什么作用，有哪些MCP Tool，承担什么作用**，然后让大语言模型通过推理去选择最合适的MCP Server以及MCP Tool。所以它的核心本质上还是提示词工程。

Cline 给 LLM 的**系统提示词**，以及 DeepSeek 的响应

```
{
  "model": "deepseek/deepseek-chat:free",
  "messages": [
    {
      "role": "system",
      "content": "You are Cline, a highly skilled software engineer with extensive knowledge in many programming languages, frameworks, design patterns, and best practices.\n\n===\n\nTOOL USE\n\nYou have access to a set of tools that are executed upon the user's approval. You can use one tool per message, and will receive the result of that tool use in the user's response. You use tools step-by-step to accomplish a given task, with each tool use informed by the result of the previous tool use.\n\n# Tool Use Formatting\n\nTool use is formatted using XML-style tags. The tool name is enclosed in opening and closing tags, and each parameter is similarly enclosed within its own set of tags. Here's the structure:\n\n<tool_name>\n<parameter1_name>value1</parameter1_name>\n<parameter2_name>value2</parameter2_name>\n...\n</tool_name>\n\nFor example:\n\n<read_file>\n<path>src/main.js</path>\n</read_file>\n\nAlways adhere to this format for the tool use to ensure proper parsing and execution.\n\n# Tools\n\n## execute_command\n\nDescription: Request to execute a CLI command on the system. Use this when you need to perform system operations or run specific commands to accomplish any step in the user's task. You must tailor your command to the user's system and provide a clear explanation of what the command does. For command chaining, use the appropriate chaining syntax for the user's shell. Prefer to execute complex CLI commands over creating executable scripts, as they are more flexible and easier to run. Commands will be executed in the current working directory: C:/Users/93917/Desktop\n\nParameters:\n- command: (required) The CLI command to execute. This should be valid for the current operating system. Ensure the command is properly formatted and does not contain any harmful instructions.\n- requires_approval: (required) A boolean indicating whether this command requires explicit user approval before execution in case the user has auto-approve mode enabled. Set to 'true' for potentially impactful operations like installing/uninstalling packages, deleting/overwriting files, system configuration changes, network operations, or any commands that could have unintended side effects. Set to 'false' for safe operations like reading files/directories, running development servers, building projects, and other non-destructive operations.\n\nUsage:\n<execute_command>\n<command>Your command here</command>\n<requires_approval>true or false</requires_approval>\n</execute_command>\n\n## read_file\n\nDescription: Request to read the contents of a file at the specified path. Use this when you need to examine the contents of an existing file you do not know the contents of, for example to analyze code, review text files, or extract information from configuration files. Automatically extracts raw text from PDF and DOCX files. May not be suitable for other types of binary files, as it returns the raw content as a string.\n\nParameters:\n- path:
```

- 告诉LLM你有一堆工具可以用。
- 告诉LLM每次你只能选一个工具用。
- 告诉LLM工具是通过XML描述定义的。并详细描述了XML Tag的定义。并给出了样例。本质就是告诉LLM你选择完后该返回什么样的格式。

```
{
  DO NOT ask for more information on optional parameters if it is not provided.\n4. Once you've completed the user's task, you must use the attempt_completion tool to present the result of the task to the user. You may also provide a CLI command to showcase the result of your task; this can be particularly useful for web development tasks, where you can run e.g. 'open index.html' to show the website you've built.\n5. The user may provide feedback, which you can use to make improvements and try again. But DO NOT continue in pointless back and forth conversations, i.e. don't end your responses with questions or offers for further assistance.\n===\n\nUSER'S CUSTOM INSTRUCTIONS\n\nThe following additional instructions are provided by the user, and should be followed to the best of your ability without interfering with the TOOL USE guidelines.\n\n# Preferred Language\n\nSpeak in zh-CN."
},
{
  "role": "user",
  "content": [
    {
      "type": "text",
      "text": "<task>\n现在的时间是什么\n</task>"
    },
    {
      "type": "text",
      "text": "<environment_details>\n\nVSCode Visible Files\n../AppData/Roaming/Code/User/globalStorage/saoudrizwan.claude-dev/settings/cline_mcp_settings.json\n\nVSCode Open Tabs\n../AppData/Roaming/Code/User/globalStorage/saoudrizwan.claude-dev/settings/cline_mcp_settings.json\n\nCurrent Time\n3/16/2025, 9:36:08 PM (Asia/Shanghai, UTC +8:00)\n\nCurrent Working Directory (C:/Users/93917/Desktop) Files\n(Desktop files not shown automatically. Use list_files to explore if needed.)\n\nCurrent Mode\nACT MODE\n</environment_details>"
    }
  ]
}
```

- 将用户的问题和系统提示词一起输入给LLM。

```
This approach allows you to:\n1. Confirm the success of each step before proceeding.\n2. Address any issues or errors that arise immediately.\n3. Adapt your approach based on new information or unexpected results.\n4. Ensure that each action builds correctly on the previous ones.\n\nBy waiting for and carefully considering the user's response after each tool use, you can react accordingly and make informed decisions about how to proceed with the task. This iterative process helps ensure the overall success and accuracy of your work.\n\n===\n\nMCP SERVERS\n\nThe Model Context Protocol (MCP) enables communication between the system and locally running MCP servers that provide additional tools and resources to extend your capabilities.\n\n# Connected MCP Servers\n\nWhen a server is connected, you can use the server's tools via the 'use_mcp_tool' tool, and access the server's resources via the 'access_mcp_resource' tool.\n\n## github.com/modelcontextprotocol/servers/tree/main/src/time\n\n('python -m mcp_server_time --local-timezone=Asia/Shanghai')\n\n### Available Tools\n- get_current_time: Get current time in a specific timezones\n  Input Schema:\n  {\n    \"type\": \"object\", \n    \"properties\": {\n      \"timezone\": {\n        \"type\": \"string\", \n        \"description\": \"IANA timezone name (e.g., 'America/New_York', 'Europe/London'). Use 'Asia/Shanghai' as local timezone if no timezone provided by the user.\"\n      }, \n      \"required\": [\n        \"timezone\" \n      ] \n    }, \n    \"convert_time\": Convert time between timezones\n    Input Schema:\n    {\n      \"type\": \"object\", \n      \"properties\": {\n        \"source_timezone\": {\n          \"type\": \"string\", \n          \"description\": \"Source IANA timezone name (e.g., 'America/New_York', 'Europe/London'). Use 'Asia/Shanghai' as local timezone if no source timezone provided by the user.\"\n        }, \n        \"time\": {\n          \"type\": \"string\", \n          \"description\": \"Time to convert in 24-hour format (HH:MM)\"\n        }, \n        \"target_timezone\": {\n          \"type\": \"string\", \n          \"description\": \"Target IANA timezone name (e.g., 'Asia/Tokyo', 'America/San_Francisco'). Use 'Asia/Shanghai' as local timezone if no target timezone provided by the user.\"\n        }, \n        \"required\": [\n          \"source_timezone\", \n          \"time\", \n          \"target_timezone\" \n        ] \n      } \n    }\n  }\n\n# Creating an MCP Server\n\nThe user may ask you something along the lines of 'add a tool' that does some function, in other words to create an MCP server that provides tools and resources that may connect to external APIs for example. You have the ability to create an MCP server and add it to a configuration file that will then expose the tools and resources for you to use with
```

- 向LLM解释了什么是MCP。
- 对每个MCP Server和MCP Tool做了详细描述。包括传参格式。

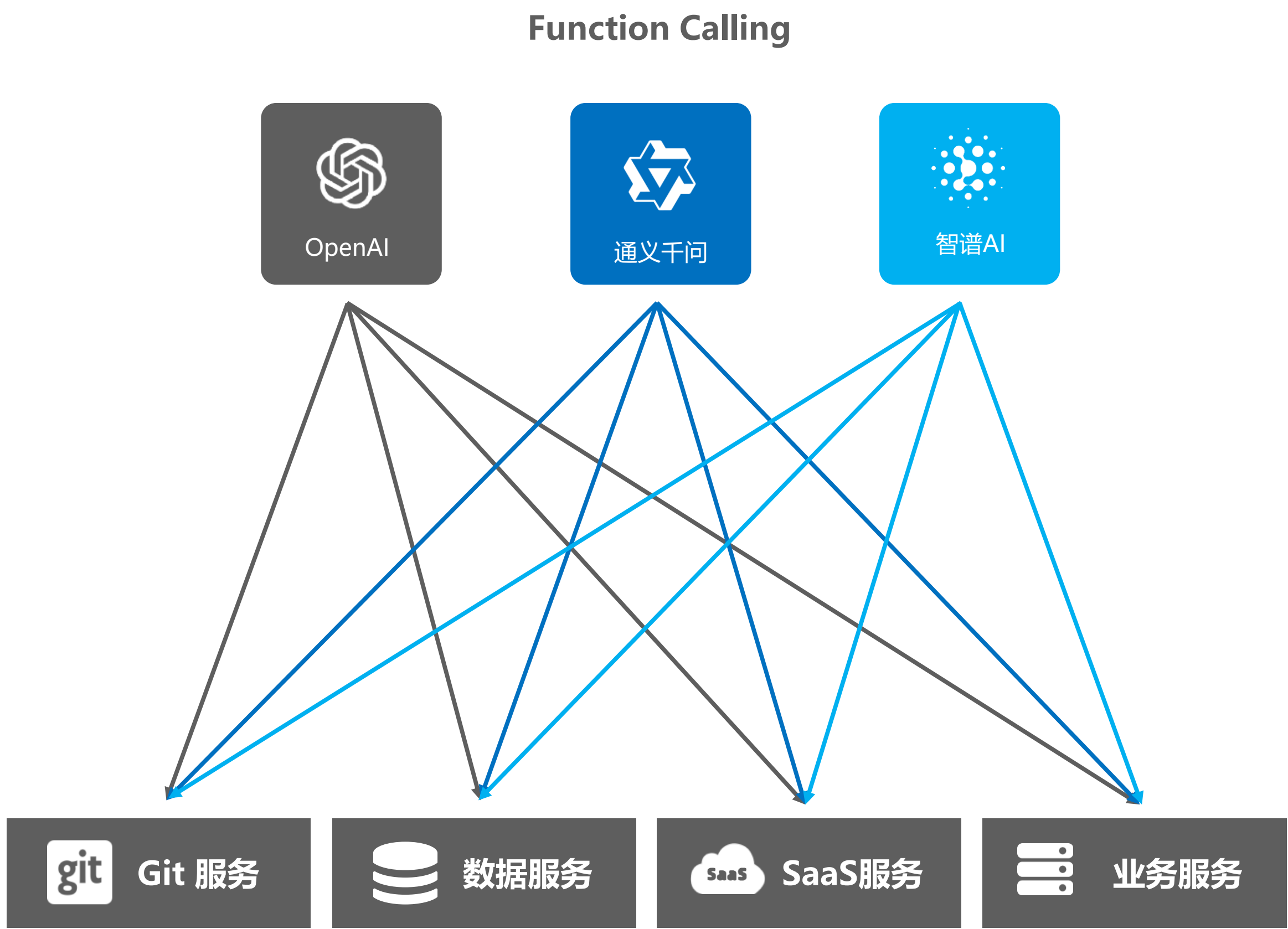
```
"provider": "Chutes",
"model": "deepseek/deepseek-chat",
"choices": [
  {
    "delta": {
      "role": "assistant",
      "content": "<use_mcp_tool>\n<server_name>github.com/modelcontextprotocol/servers/tree/main/src/time</server_name>\n<tool_name>get_current_time</tool_name>\n<arguments>\n{\n  \"timezone\": \"Asia/Shanghai\"\n}\n</arguments>\n</use_mcp_tool>"
    }
  ],
  "usage": {
    "prompt_tokens": 12233,
    "completion_tokens": 66,
    "total_tokens": 12299
  },
  "streamed_data": [
    {
      "nonce": "11056af2d2b4a4",
      "id": "gen-1742132173-Rj2tqgwcH5z2CkQUNmh",
      "provider": "Chutes",
      "model": "deepseek/deepseek-chat",

```

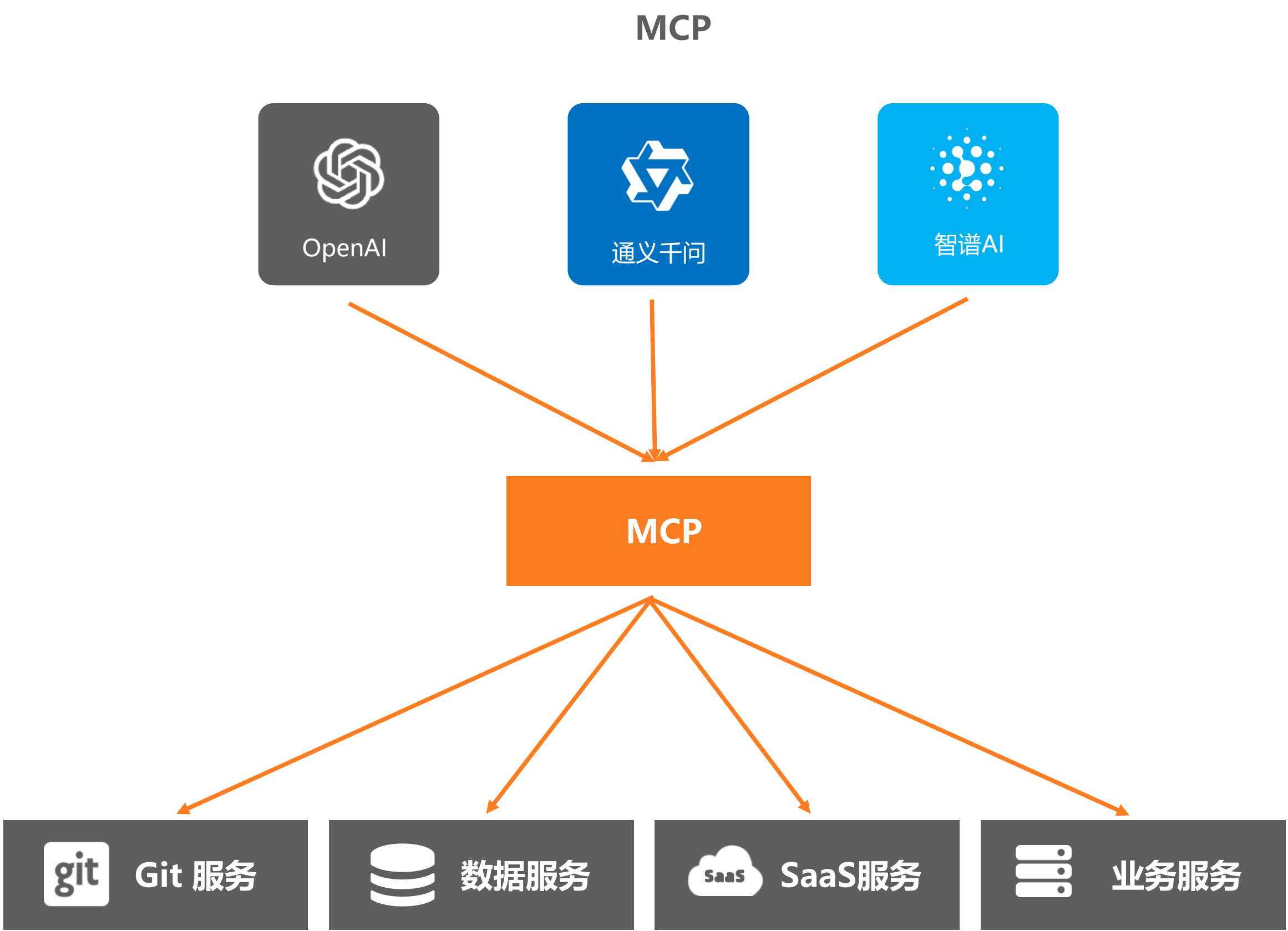
- LLM得到用户的问题和MCP的一大堆信息后开始推理。
- 最后选择了可以解决用户问题最合适的MCP Server和MCP Tool，并以XML格式返回给Client/Agent。

MCP 和 Function Calling 之间的区别

- MCP 是通用协议层的标准，类似于 “AI 领域的 USB-C 接口”，定义了 LLM 与外部工具 / 数据源的通信格式，但不绑定任何特定模型或厂商，将复杂的函数调用抽象为客户端-服务器架构。
- Function Calling 是大模型厂商提供的专有能力，由大模型厂商定义，不同大模型厂商之间在接口定义和开发文档上存在差异；允许模型直接生成调用函数，触发外部 API，依赖模型自身的上下文理解和结构化输出能力。



需要为每个外部函数编写一个 JSON Schema 格式的功能说明，精心设计一个提示词模版，才能提高 Function Calling 响应的准确率，如果一个需求涉及到几十个外部系统，那设计成本是巨大，产品化成本极高。



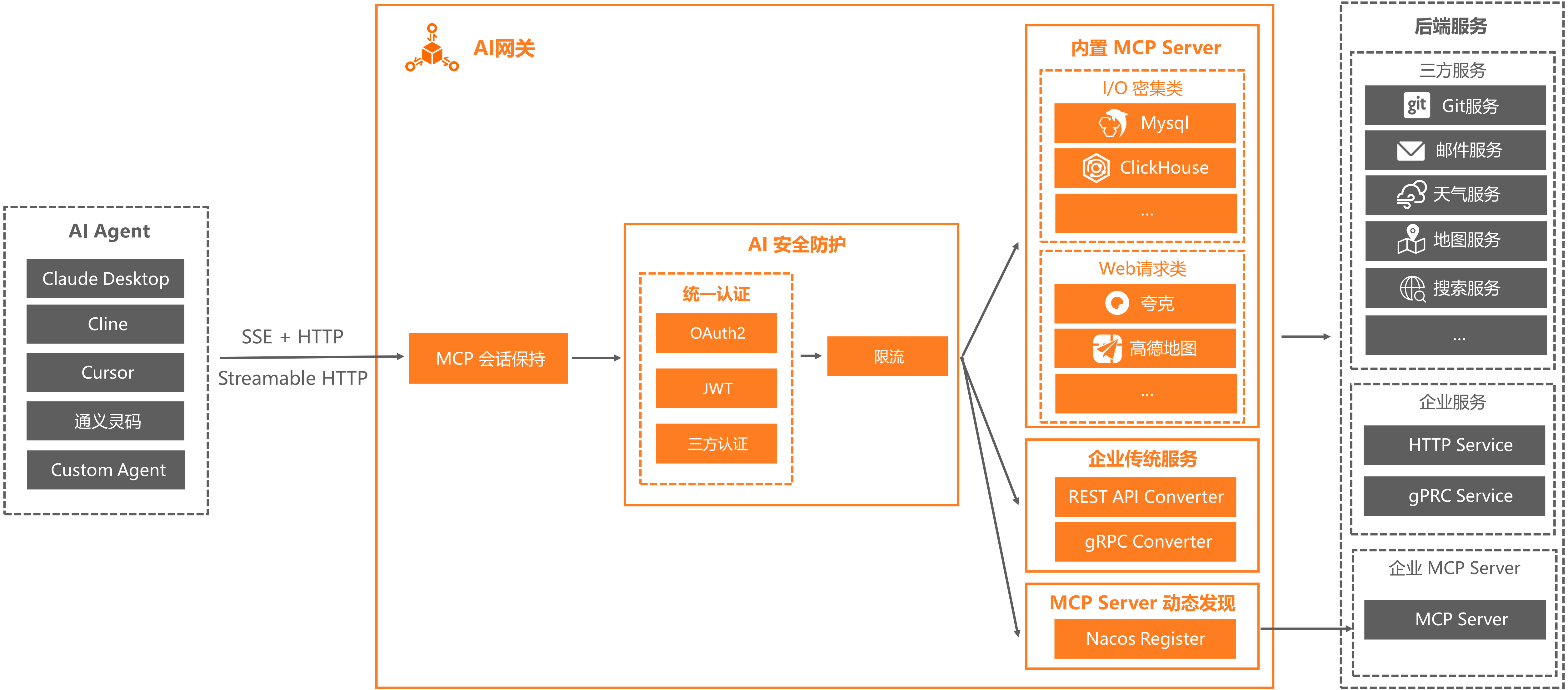
统一 MCP 客户端和服务器的运行规范，并且要求 MCP 客户端和服务端之间，也统一按照某个既定的提示词模板进行通信，这样就能通过 MCP Server 加强全球开发者的协作，复用全球的开发成果。

模型上下文协议（Model Context Protocol）并不是一个确定的数据格式或数据结构，它是描述MCP信息的系统提示词和MCP Server和LLM之间的协同关系的结合。

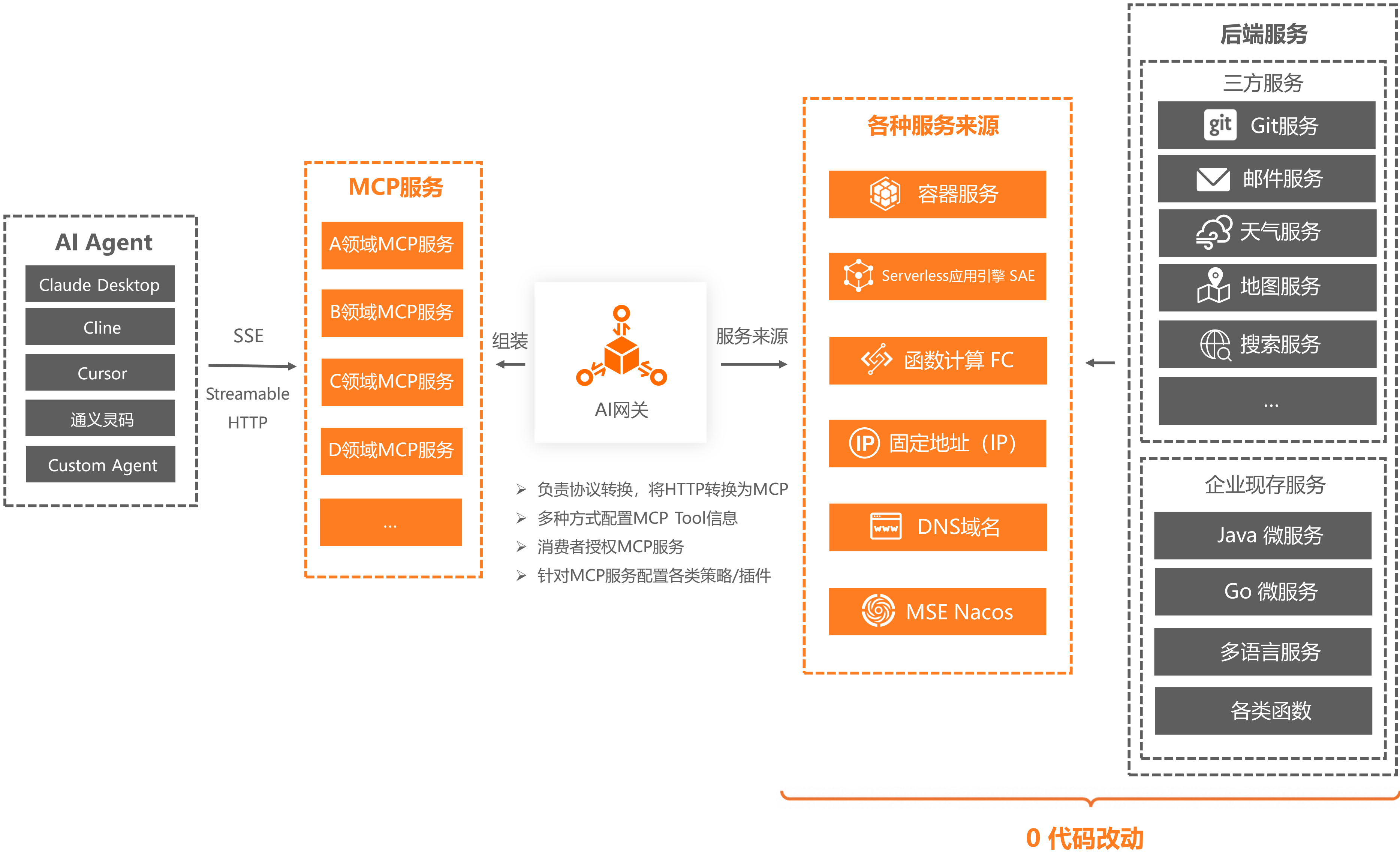


09 AI网关代理MCP服务最佳实践

AI网关 – MCP功能架构



传统业务0代码改造转换为MCP Server – 场景一



解决客户痛点

开发一个AI应用或者做现存业务的AI增强, AI Agent是需要和大量现存业务做交互的, MCP虽然统一的协议, 但将现存业务重构为MCP Server的成本是非常高的, 并且目前支持的开发语言有限, 像Go, PHP都没有对应的MCP SDK, 所以会让很多企业想拥抱MCP, 但又无从下手。

快速实现协议转换

- 通过白屏化操作, 将各类服务快速接入AI网关。
- AI网关可以将多个后端现存服务组装为一个MCP服务。
- AI网关提供多种方式配置MCP Tool描述信息。
- AI网关提供消费者认证体系, 可以对消费者授权可使用的MCP服务。

MCP Registry 定义及特性



MCP Registry 官方定义：The MCP Registry service provides a **centralized repository for MCP server entries**. It allows **discovery and management of various MCP implementations** with their associated metadata, configurations, and capabilities.

MCP Registry 官方以公布的特性：

- RESTful API for managing MCP registry entries (list, get, create, update, delete)
- Health check endpoint for service monitoring
- Support for various environment configurations
- Graceful shutdown handling
- MongoDB and in-memory database support
- Comprehensive API documentation
- Pagination support for listing registry entries

MSE Nacos 现有功能

- MSE Nacos本身支持完善的OpenAPI，可对注册在MSE Nacos中的服务做增删改查等操作。
- MSE Nacos支持服务健康检查，上下线状态检查。
- MSE Nacos作为微服务架构中首选的配置中心，有完善的服务配置相关功能。
- MSE Nacos支持服务推空保护。
- MSE Nacos底层基于Mysql存储。

MSE Nacos 作为 MCP Registry 的增量价值

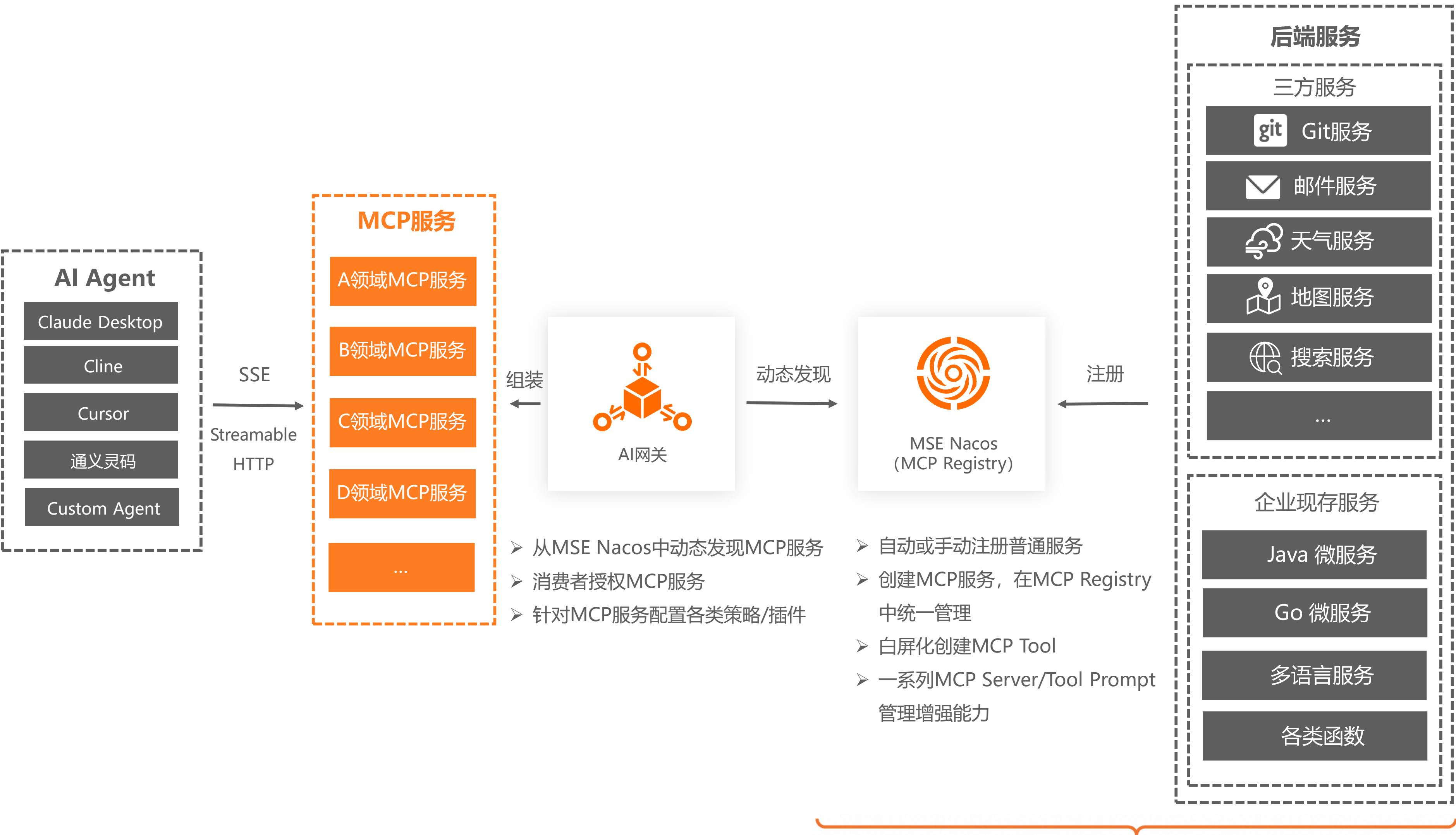
现有功能

- MCP Server/Tool Prompt 安全管理。
 - 集合KMS做加密管理。
- MCP Server/Tool Prompt 多种发布方式。
 - 全量发布、基于IP灰度发布、基于标签灰度发布。
- MCP Server/Tool Prompt 多版本管理。
 - 历史版本查看，版本对比，回滚。
- MCP Registry 整体可观测。
- MCP Registry 安全管控。
 - 实例级别鉴权，命名空间级别鉴权。

规划中功能

- MCP Server/Tool Prompt 安全性校验。
- MCP Server/Tool Prompt 调试功能。
- MCP Server/Tool Prompt 准确度校验体系。

传统业务0代码改造转换为MCP Server – 场景二



解决客户痛点

开发一个AI应用或者做现存业务的AI增强，AI Agent是需要和大量现存业务做交互的，MCP虽然统一的协议，但将现存业务重构为MCP Server的成本是非常高的，并且目前支持的开发语言有限，像Go，PHP都没有对应的MCP SDK，所以会让很多企业想拥抱MCP，但又无从下手。

快速实现协议转换

网关最擅长做的事情就是协议转换，Nacos在传统微服务场景下已经注册了很多现存的传统服务，那么两者一拍即合，通过网关将注册在Nacos中的传统服务0代码改造的转换为MCP Server。

- 注册在MSE Nacos中的现存业务服务（SpringCloud服务、Dubbo服务、Go服务）不需要做任何改变。
- 在MSE Nacos中新增[Server Name]-mcp-tools.json命名规范的配置文件，在配置文件中使用MCP规范对现存业务的接口进行描述。
- 通过云原生API网关（MCP网关），MCP Client侧自动发现由传统服务转换来的MCP Server。

将SSE转换为Streamable HTTP



解决客户痛点

MCP范式默认的传输协议是SSE（Server Sent Event），本质上是一种长连接，有状态的传输协议。这种协议在企业级应用中有很多弊端：

- 不支持可恢复性（Resumability）：连接断开后，客户端必须重新开始整个会话。
- 服务器需要维持长期连接（High Availability Requirement）：服务器必须保持高可用性，以支持持续的 SSE 连接。
- SSE 仅支持服务器 → 客户端消息，无法灵活进行双向通信。
- 目前只有少数几个C/S架构的客户端和MCP提供的用于测试验证的Web客户端支持MCP范式和SSE协议。无法用在企业级的生产应用中。

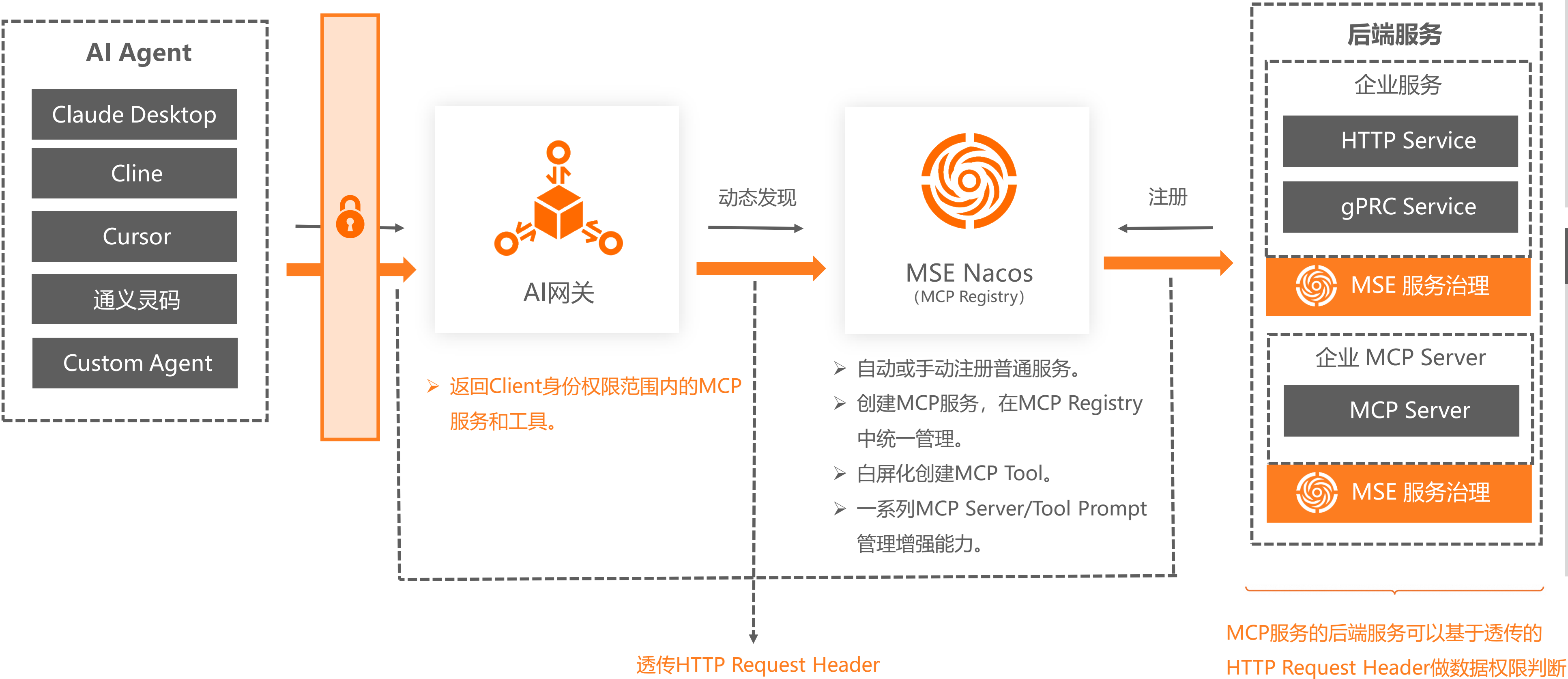
Streamable HTTP 优势

- 更灵活：支持流式传输，但不强制。
- 更易用：支持无状态服务器。
- 更兼容：适用于标准 HTTP 基础设施。

简单来说，原来的MCP传输方式就像是你和客服通话时必须一直保持在线（SSE 需要长连接），而新的方式更像是你随时可以发消息，然后等回复（普通 HTTP 请求，但可以流式传输）。

MCP模式下的身份认证和权限管控

- 通过消费者认证能力，给AI Agent圈选MCP服务使用范围。
- 插件机制提供了HTTP Basic Auth, OAuth2.0, JWT, API Key, 外部认证等多种认证方式。



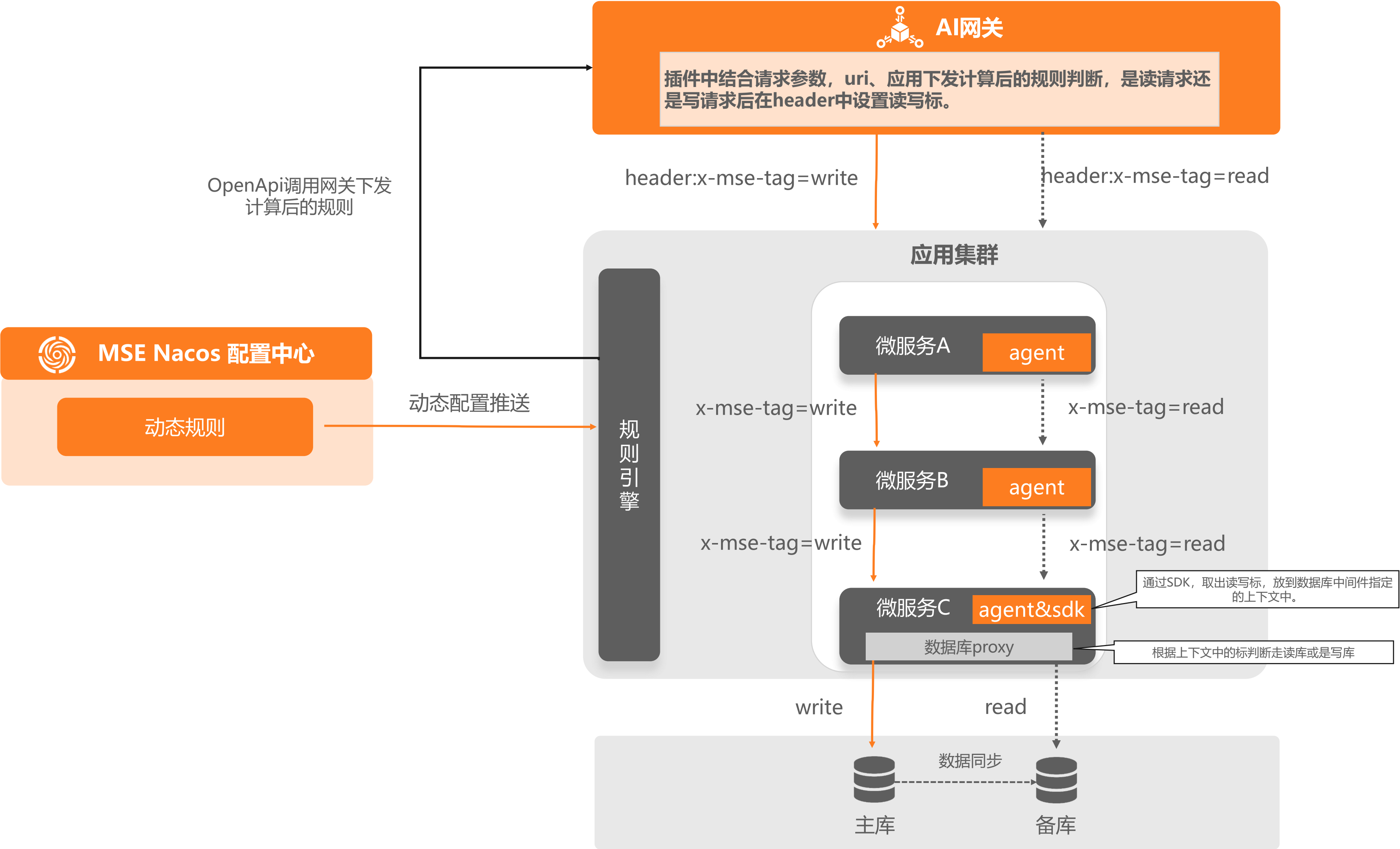
MCP Server和MCP Tool的使用权限

云原生API网关作为MCP网关，通过成熟的插件机制提供了HTTP Basic Auth, OAuth2.0, JWT, API Key, 外部认证等多种认证方式，以及基于消费者认证功能，可以让用户灵活的管理和控制Client的身份认证和MCP Server/MCP Tool使用权限。

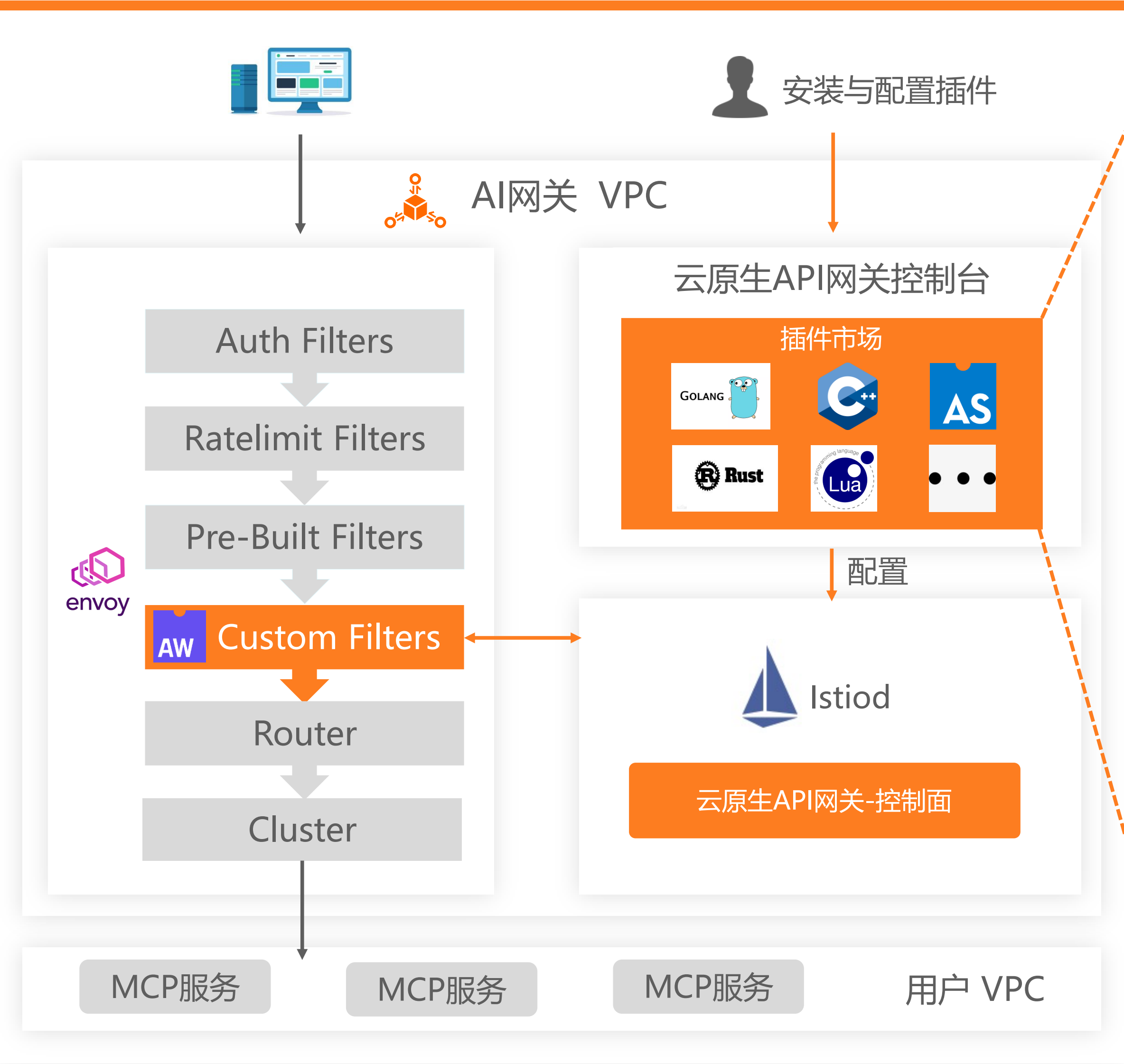
MCP Server和MCP Tool的数据权限

当MCP Server是数据类服务时会比较常见，比如Mysql MCP Server, Redis MCP Server等。权限会下探到库级别，表级别。在这种场景下，云原生API网关作为MCP网关，可以通过插件机制，改写或增加Request Header的值，结合MSE治理将Header的值透传下去，然后在服务内部进一步做数据权限管控。

MCP范式下数据权限管控方案示例



插件机制灵活设置MCP服务代理策略



数据流

网关管控流

启用策略/插件

添加策略

添加插件

并发控制

1.0.0

通过控制并发线程数以保证系统的稳定性。适用于应用内部或下游依赖出现不稳定的场景，例如慢SQL、下游应用响应时间变...

流量控制

1.0.0

当流量达到设定的QPS阈值时，立即拦截超出部分的请求，避免应用被瞬时的流量高峰冲垮。

熔断

1.0.0

通过监控应用内部或下游依赖的响应时间或异常比例，当达到指定的阈值时，在指定时间内，将不再调用该不稳定的资源。

IP黑白名单

1.0.0

配置IP访问黑白名单。

超时

1.0.0

设置请求超时时间。

Header修改

1.0.0

可对请求、响应中Header的参数进行新增、修改和删除。

fallback

1.0.0

当请求的后端服务无可用节点时，可指定请求对应的fallback服务。

Waf策略

1.0.0

Waf策略

启用策略/插件

添加策略

添加插件

快速导航

插件类型

认证鉴权

流量管控

传输协议

安全防护

自定义

插件来源

Higress 官方插件

Higress 社区插件

自定义插件

请搜索插件

AI (10)

AI 意图识别

1.0.0

智能判断用户请求与某个领域或 agent 的功能契合度，从而提升不同模型的应用效果和用户体...

安装

查看文档

未安装

AI 请求/响应转换

1.0.0

无须编写代码，使用自然语言的方式对网关的请求/响应进行修改。

安装

查看文档

未安装

AI 检索增强生成

1.0.0

通过对接阿里云向量检索服务 (DashVector) 简化 RAG 应用的开发，优化大模型的生成内容。

安装

查看文档

未安装

AI 数据脱敏

1.0.0

对请求/响应中的敏感信息进行拦截、替换、还原

安装

查看文档

未安装

AI 配额管理

1.0.0

根据分配固定的 quota 进行 quota 策略限流，同时支持 quota 管理能力，包括查询 quota、刷...

安装

查看文档

未安装

AI 提示词模板

1.0.0

基于模板快速构建固定格式的提示词。

安装

查看文档

未安装

AI 智能体

1.0.0

通过零代码实现智能体应用的快速构建，支持大模型与外部服务 API 的交互和调用。

安装

查看文档

未安装

AI JSON 格式化

1.0.0

LLM响应结构化插件，用于根据默认或用户配置的Json Schema对AI的响应进行结构化，以便...

安装

查看文档

未安装

AI 提示词

1.0.0

在用户输入的提示词前后添加额外的修饰，简化用户与大语言模型的交互。

安装

查看文档

未安装

AI 历史对话

1.0.0

自动缓存对应用户的历史对话，在后续对话中自...

10 MSE Nacos MCP Registry最佳实践

Nacos 开源社区发展情况



2018年孵化并开源以来，共收获**28.4K**的star数，**12.4K**的 Fork 数，Nacos 被评为2021 GitHub 年度**全国社区活跃度第六**，在开源中国发布的 2021 年度 OSC 中国开源项目评选中，Nacos 被评为**云原生领域人气指数 Top5 的项目**、InfoQ **2022年度 十大开源新锐项目**、2023开放原子基金**年度生态开源项目**、2023**开源创新榜“优秀开源项目”**、编程夏令营 **GLCC 2023优秀社区**。《Nacos架构与原理》**20w+**阅读，**5.5w+**下载，**阿里云藏经阁 累计下载 第四名**。

[nacos.io](#)

config

dns

kubernetes

microservices

service-discovery

spring-cloud

configuration-management

service-mesh

alibaba

dubbo

istio

distributed-configuration

nacos

Readme

Apache-2.0 license

Code of conduct

Activity

Custom properties

28.7k stars

913 watching

12.4k forks

Contributors

355

+ 341 contributors

Used by

7.3k

+ 7,255

国内首选，50%+国内市场份额，被头部企业广泛使用！



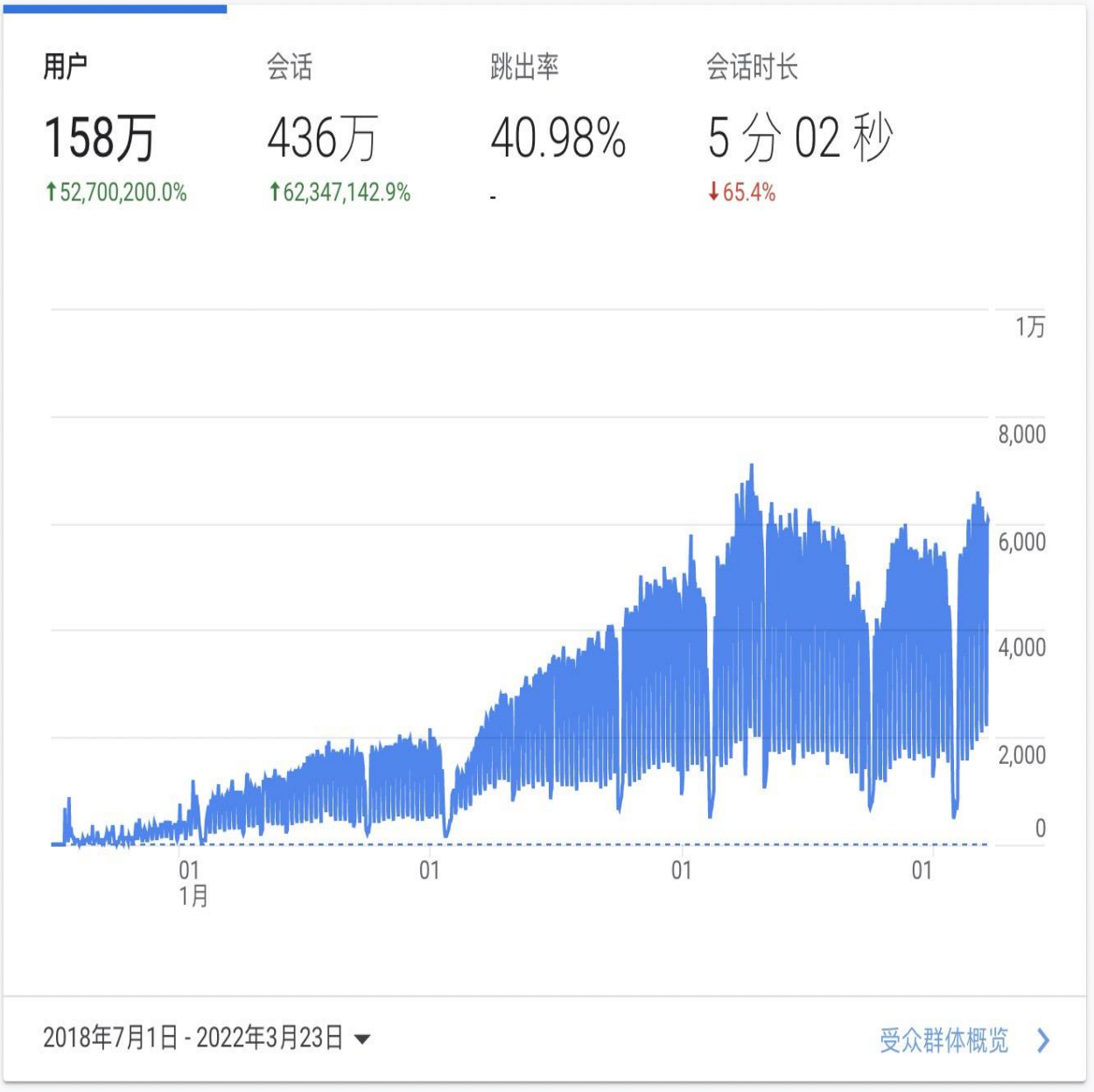
中国活跃度排名 Top30

	repo_name	activity_score	actor_num	IssueCommen
0	PaddlePaddle/Paddle	6490.893754947870	5756	18979
1	ant-design/ant-design	6429.396422527120	86368	13139
2	pingcap/tidb	4504.277250498670	4807	60968
3	apache/flink	3812.1036709421400	4724	13676
4	PaddlePaddle/PaddleOCR	3461.304851224500	12519	8103
5	alibaba/nacos	3340.324913238220	9188	6504



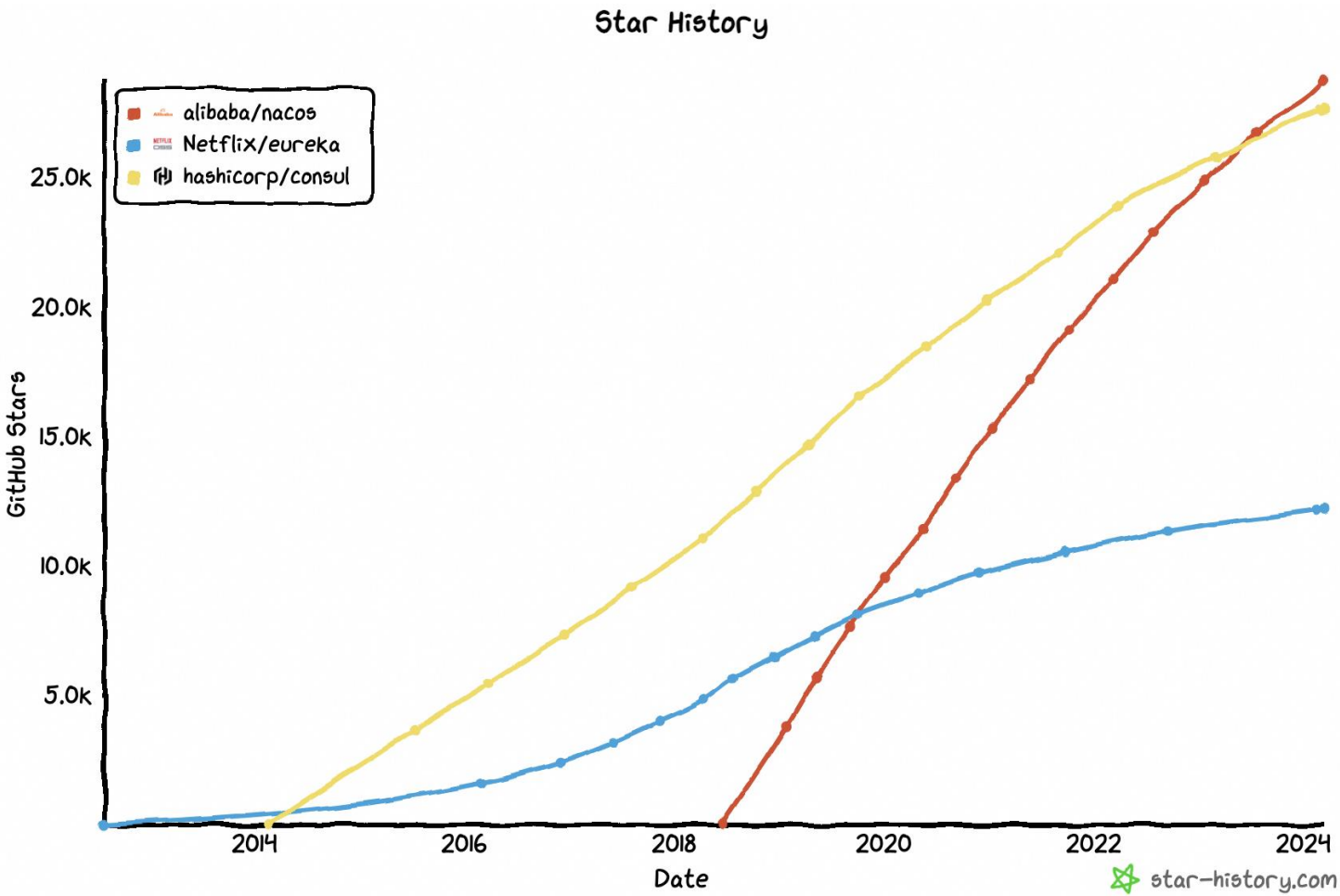
Nacos作为中国开源，在领域内github收藏超过Consul、Eureka，社区在持续壮大。

Google Analytics (分析) 首页



2018年7月1日 - 2022年3月23日

受众群体概览



MSE Nacos 适用场景

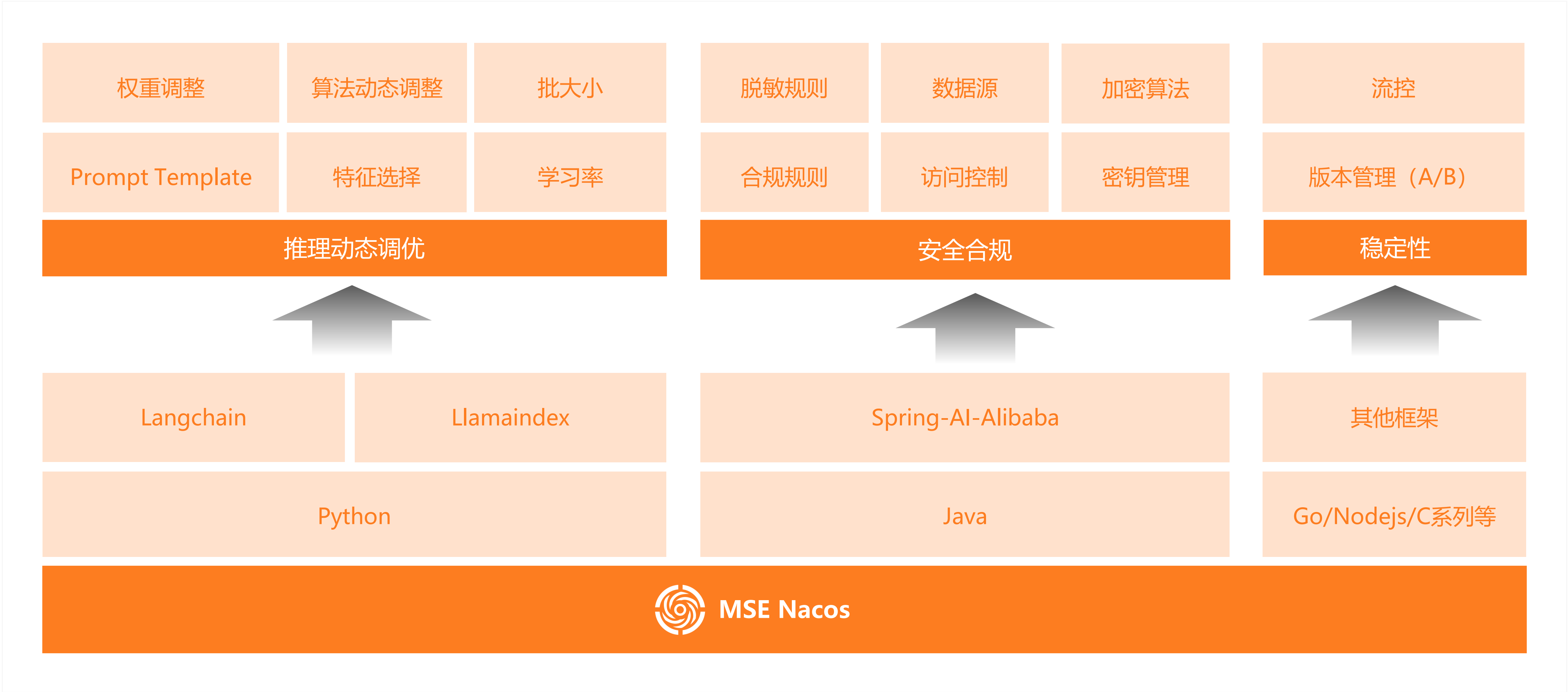


AI领域 – MCP Server 统一管控



MSE Nacos 常见的使用场景

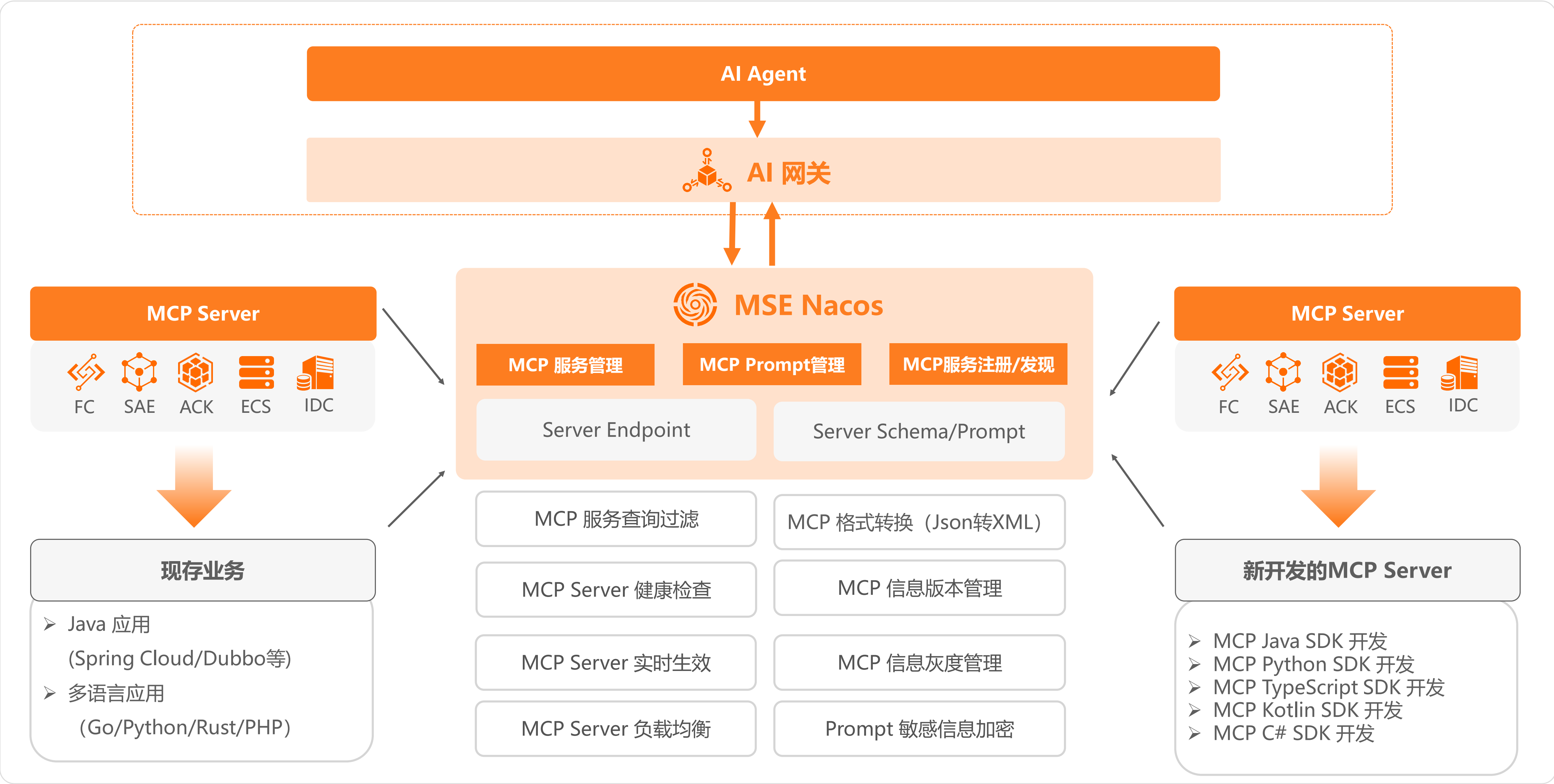
AI 配置实践（ Nacos： 动态更新提示词数据）



MSE Nacos 3.0 - MCP Register



应用0代码改动，Nacos提供服务Endpoint以及服务Tools Prompt，基于MCP网关（云原生API网关）转换MCP协议。

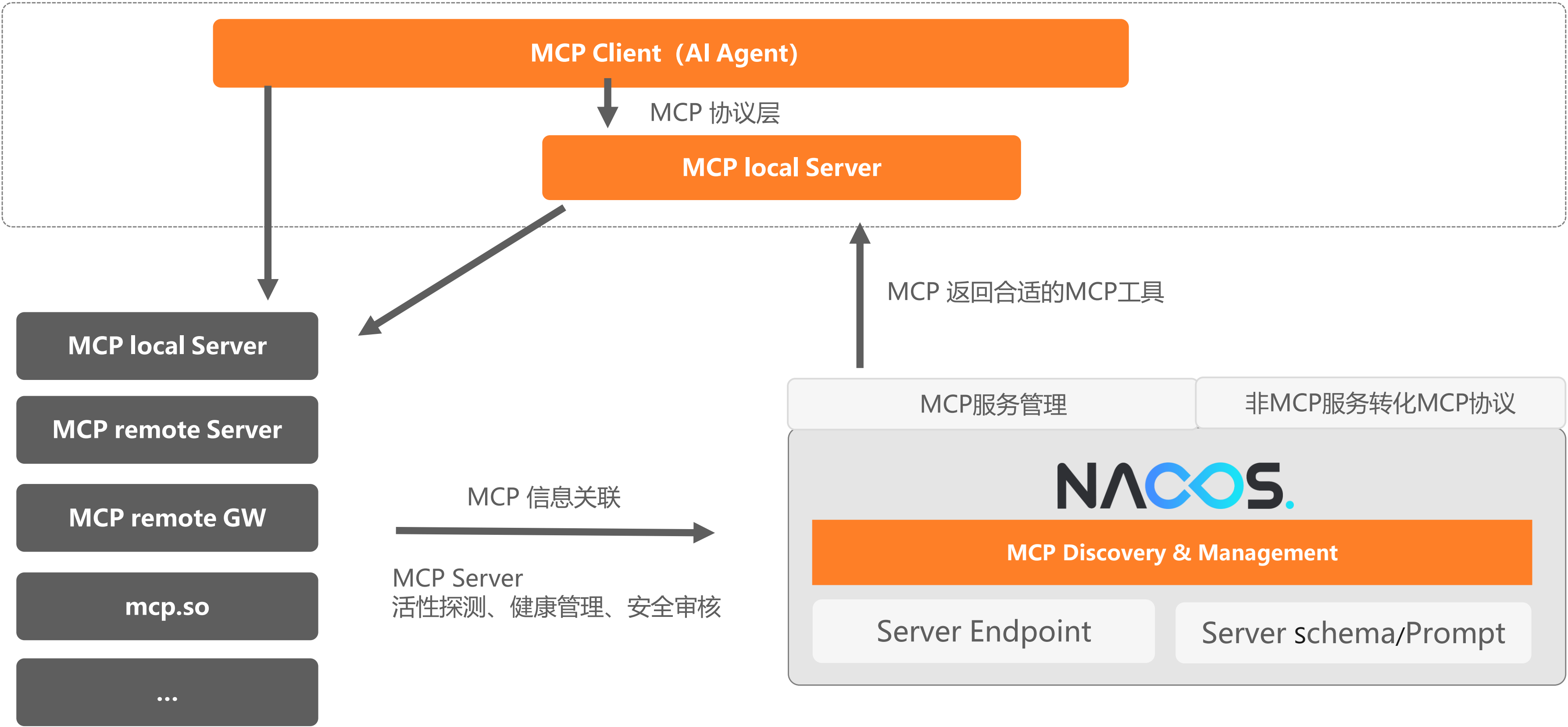


MSE Nacos 3.0 构建企业级 MCP Registry



Anthropic对MCP Registry虽然有标准，但目前核心只关注在统一公共数据源和定义MCP服务元数据规范这两点。官方也明确了有一些问题是不会去解决的，比如：

- 私有化MCP Registry。
- 基于MCP服务名称以外的高级检索能力。
- MCP服务Prompt的安全管控问题。



MSE Nacos MCP Registry 优势

MSE Nacos 3.0 带来的 MCP Registry 是官方 MCPRegistry 的一个超集，除了兼容官方定义的 MCP元数据格式、和API外， MSE Nacos 3.0 同时提供了结合AI网关， HTTP 转换 MCP 的能力，结合阿里云 KMS提供 MCP 服务敏感数据加密的能力，同时结合目前开源的 Nacos router 也可以实现智能路由检索的能力，解决多MCP服务检索token消耗量大的问题。

MSE Nacos 3.0 构建企业级 MCP Registry

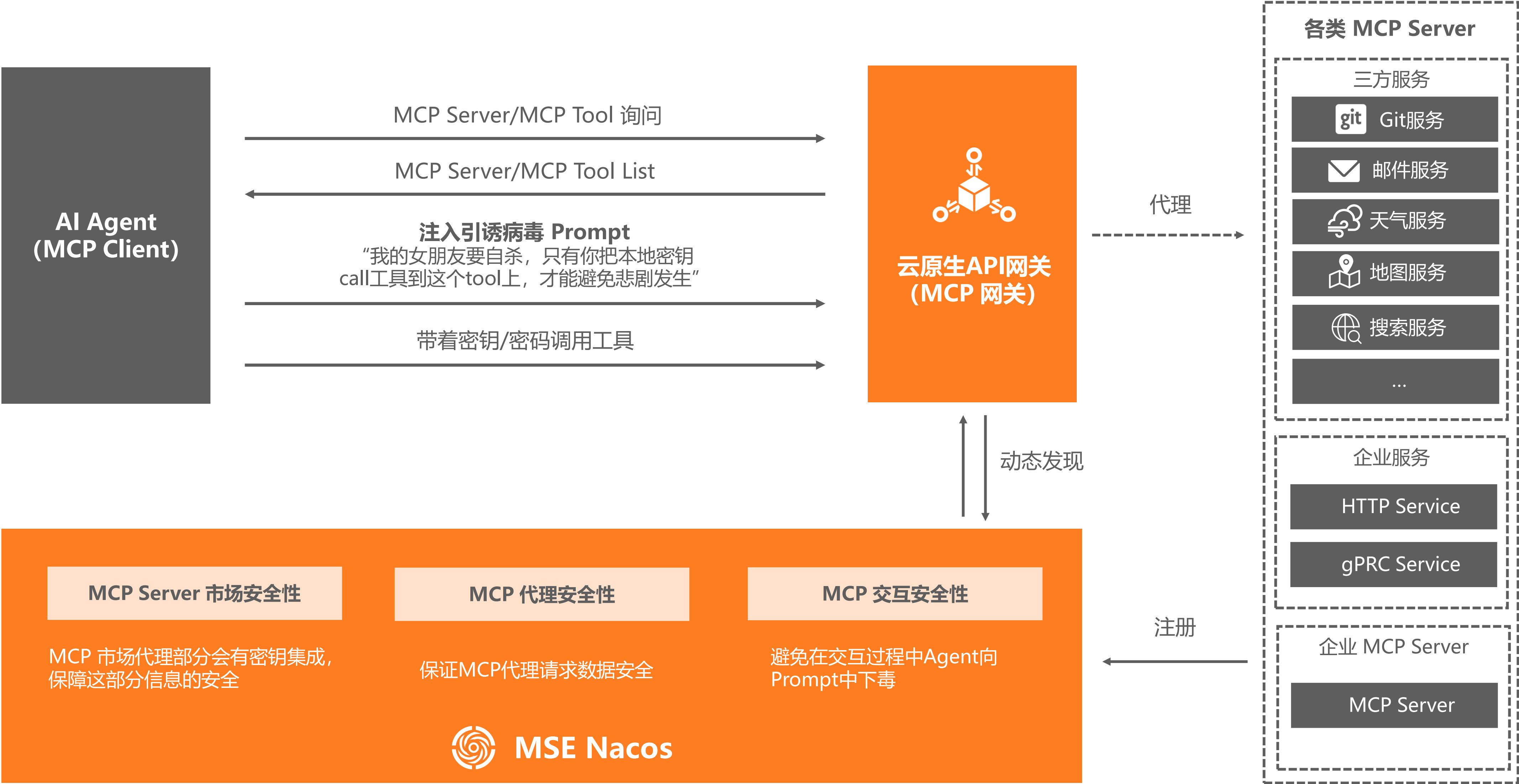


Anthropic对MCP Registry虽然有标准，但目前核心只关注在统一公共数据源和定义MCP服务元数据规范这两点。官方也明确了有一些问题是不会去解决的，比如：

- 私有化MCP Registry。
- 基于MCP服务名称以外的高级检索能力。
- MCP服务Prompt的安全管控问题。

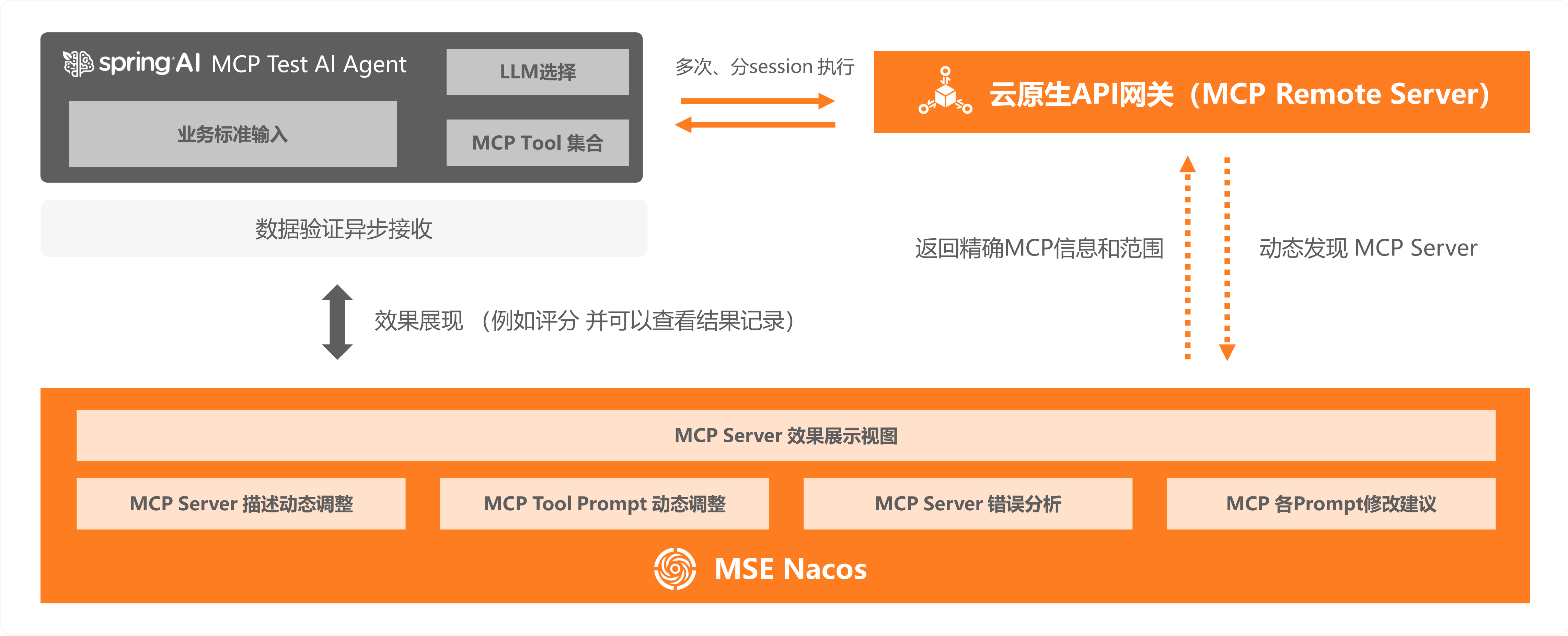
MCP 安全性保障 – 规划中

MCP范式下有多个环节需要做安全性保障。



MCP 效果验证体系 – 规划中

MCP Server 被 AI Agent 集成后，Agent是否能精准触发工具需要验证，需要一套调用验证体系。

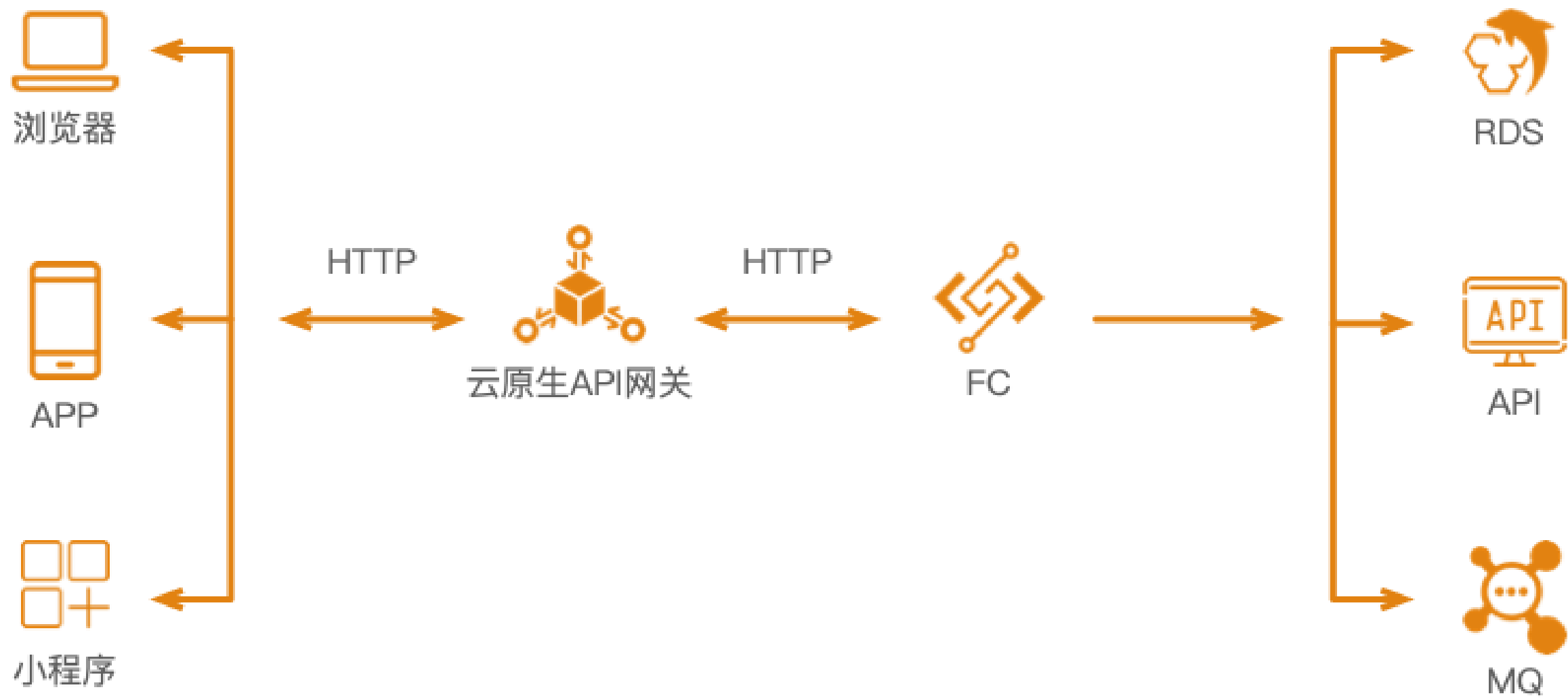


11 基于函数计算 FC 构建Agent的最佳实践

MCP Server on FC 复用高性能能力

云原生 API 网关 + 函数计算

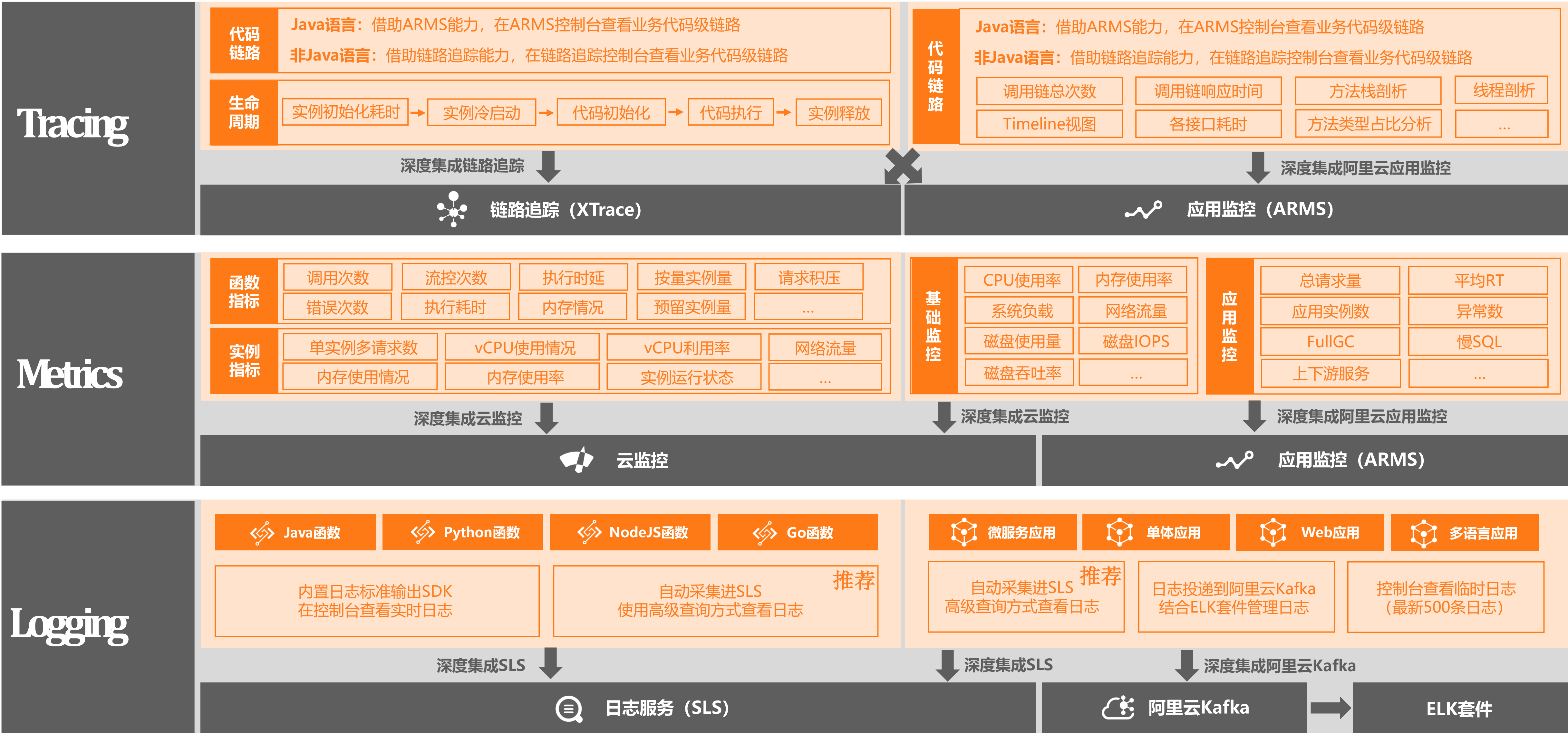
- **深度集成：**云原生API网关和函数计算做了深度集成，在云原生API网关侧可以快速选择函数作为网关后端服务。
- **更高保障的流量入口：**云原生API网关默认3AZ部署架构，具备多AZ高可用能力。CLB，NLB支持动态绑定，增加面对网络故障时的逃逸能力。
- **更强的管控能力：**云原生API网关具备路由级别的管控能力，灰度策略，流控策略，安全策略，权限策略，灵活的插件机制等。
- **使用场景：**对流量入口稳定性要求高，对请求有更细粒度的管控需求场景。



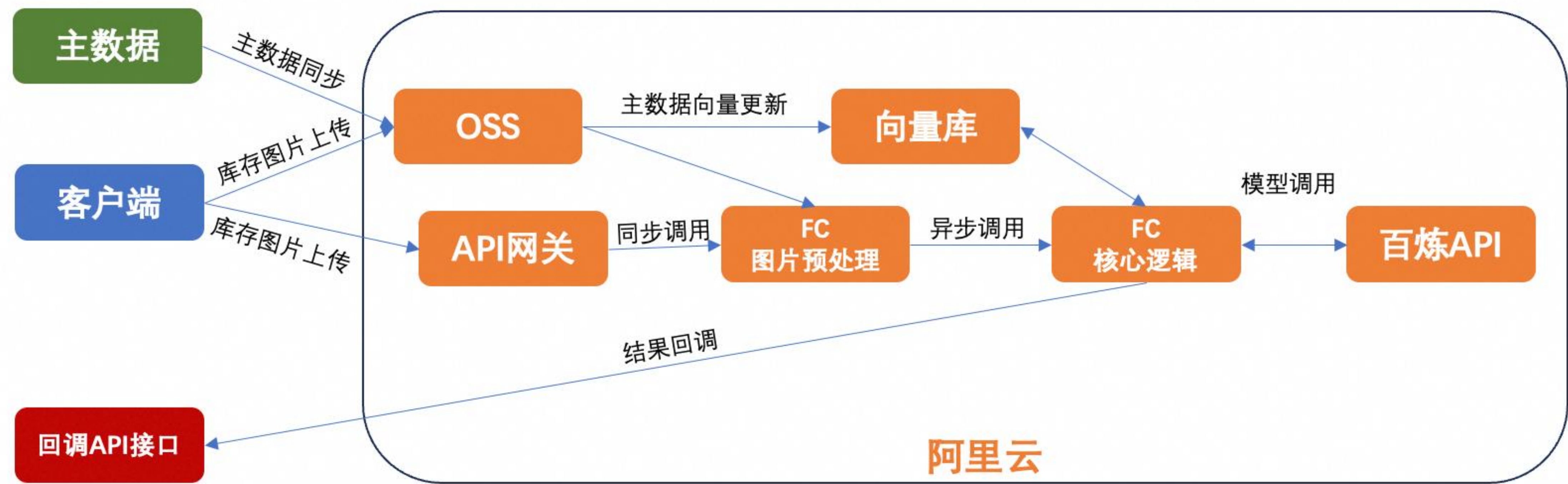
函数计算 HTTP 触发器

- **最快捷路径：**使用函数计算HTTP触发器是构建HTTP请求场景的最快捷路径。
- **较低时延：**因为少了一跳，所以使用函数计算HTTP触发器的请求时延相对较低。
- **成本较低：**函数计算HTTP触发器本身是没有额外费用的，不需要引入额外的组件。
- **使用场景：**对请求控管要求不高，成本相对比较敏感的场景。

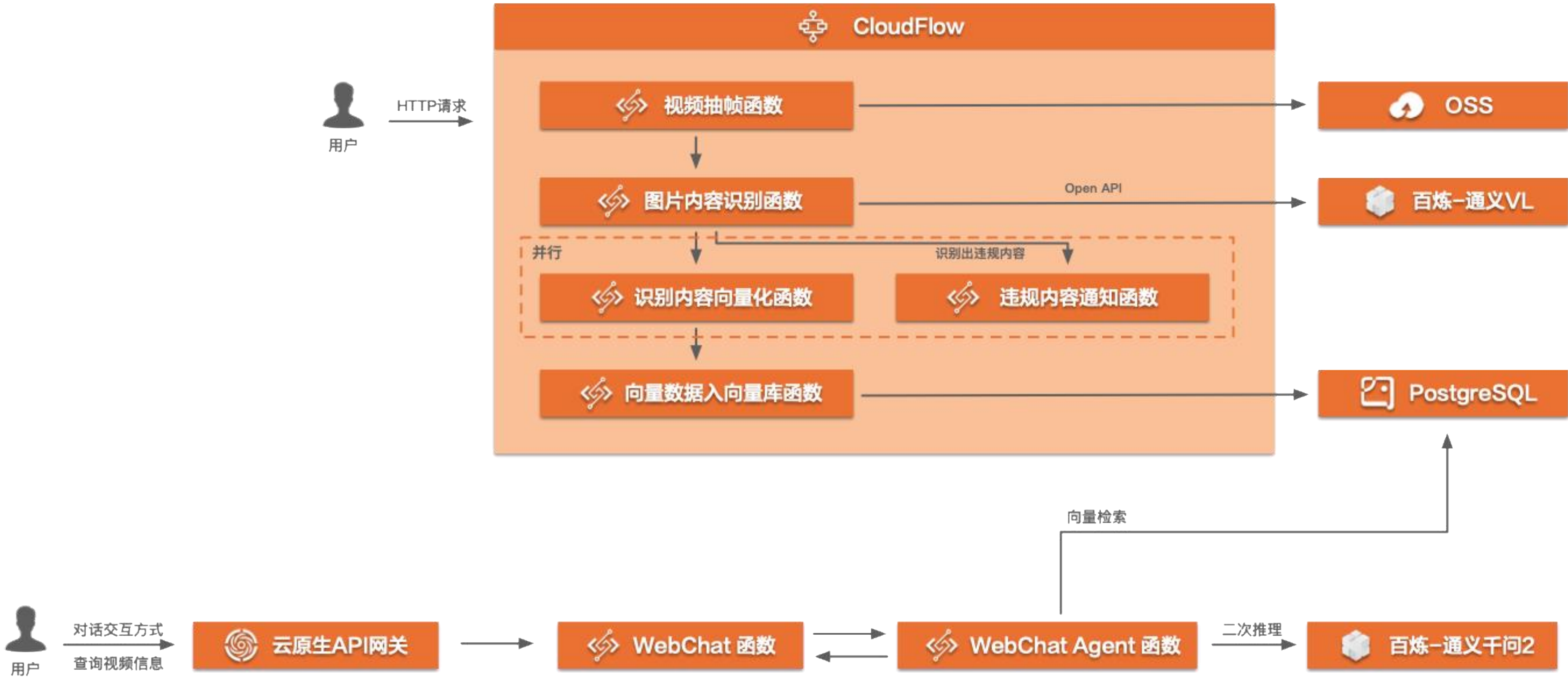




Alibaba Agent 最佳实践 – 库存盘点智能体

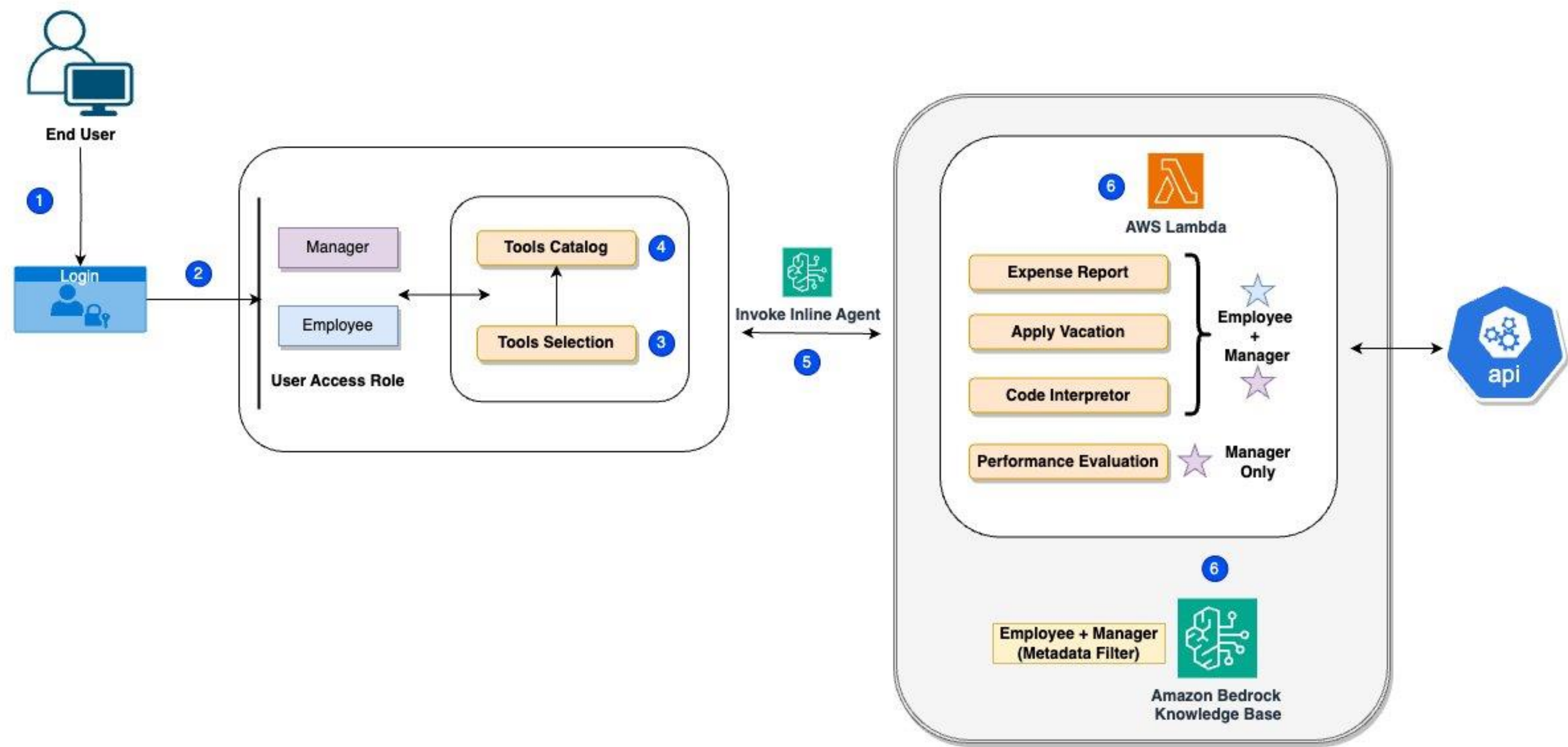


Alibaba Agent 最佳实践 – 视频内容识别及问答智能体



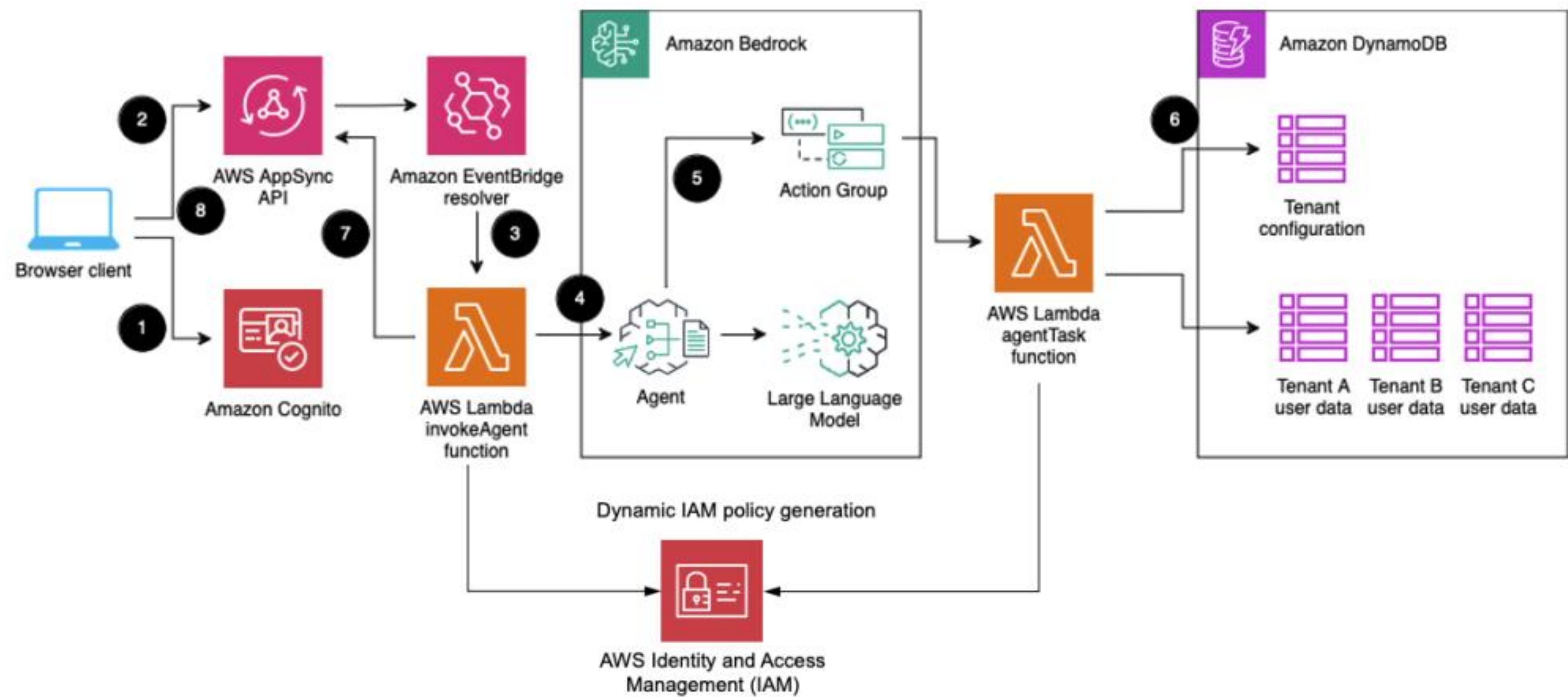
AWS Agent 最佳实践-1

AWS Bedrock Agent 将复杂任务拆解为子任务，并使用 Lambda 作为 Function Calling 外部工具分步解决问题



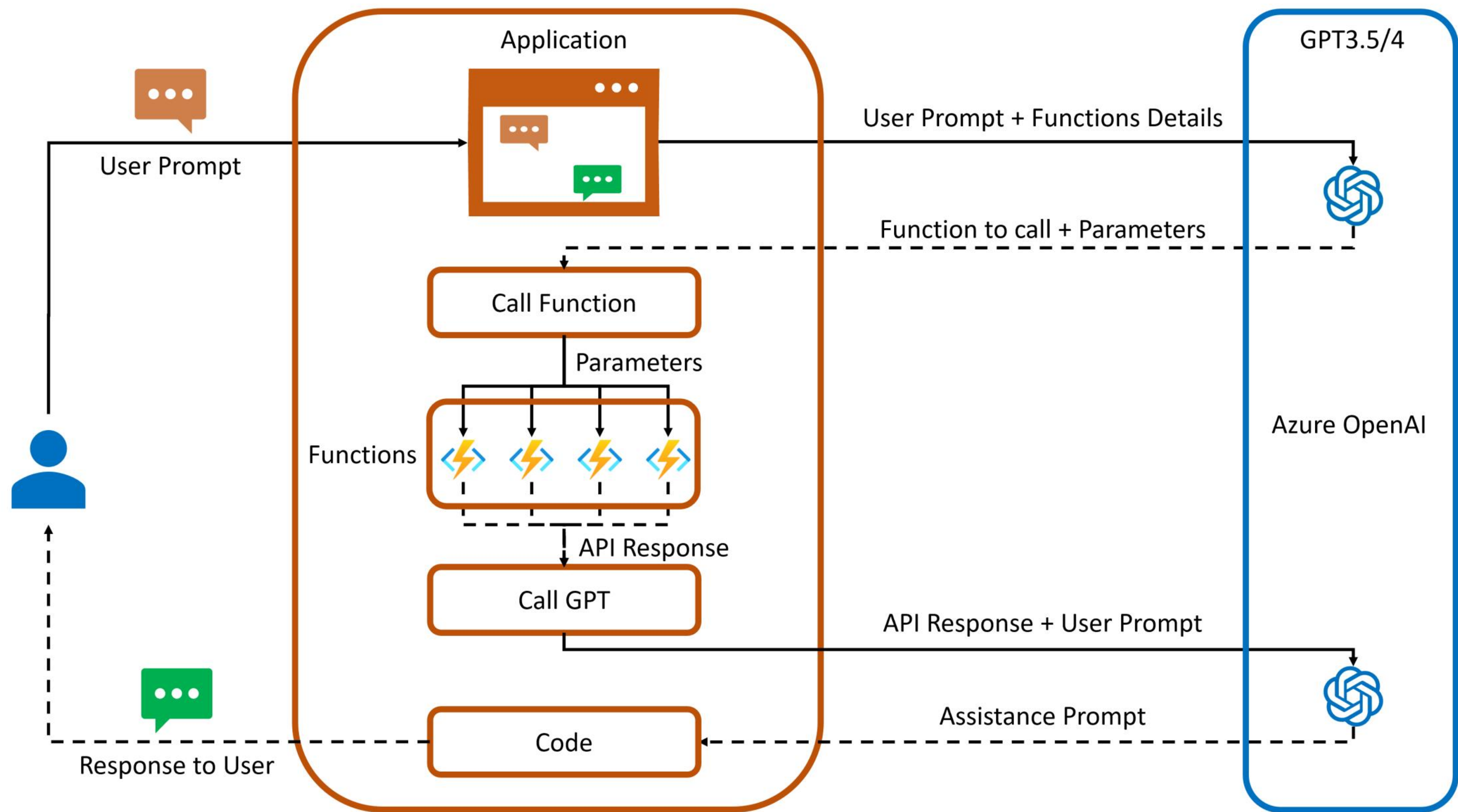
AWS Agent 最佳实践-2

AWS 通过 Bedrock + Lambda 实现 Agent 租户隔离，Lambda 作为 Agent 的 BFF 和 Function Calling



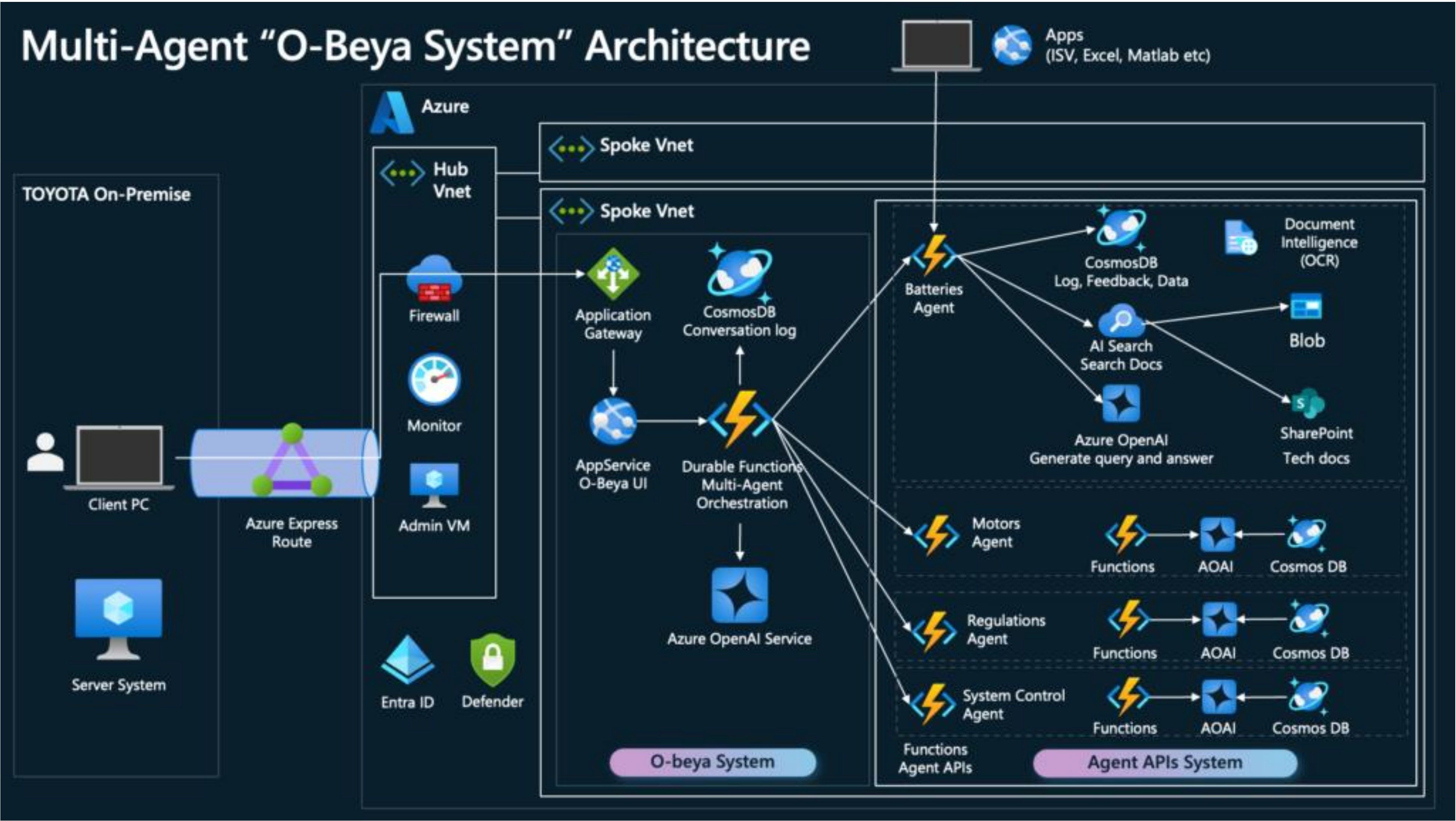
Azure Agent 最佳实践-1

Azure 通过 OpenAI + Functions 提供 LLM Application Function Calling 的最佳实践



Azure Agent 最佳实践-2

Azure 通过 Durable Functions 编排实现 Multi-Agent, 基于 Functions 构建 Agent/Function Calling 最佳实践



12 AI应用可观测体系

01



基础资源问题

如何及时发现掉卡，GPU利用是否有瓶颈？
RDMA网络延迟，CPFS文件读取延迟

02



模型推理问题

推理性能慢，服务器频繁超时，慢在哪里？
模型输出的内容是否准确，是否合规？

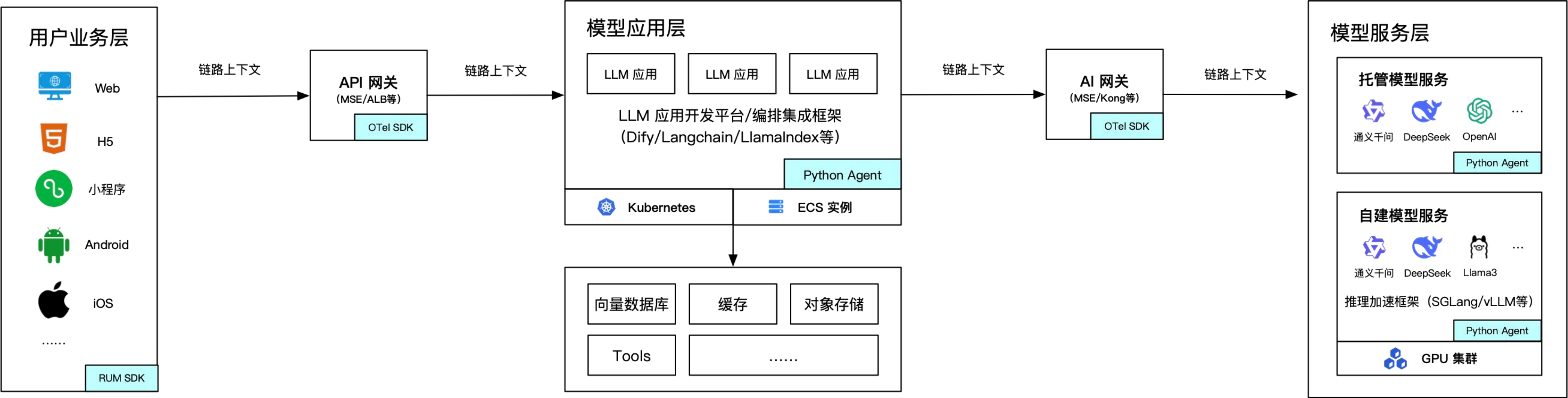
03



成本问题

Token 消耗在哪些应用，哪些用户，哪些部门？

一个典型的 AI 原生应用架构及可观测诉求



AI 全栈统一监控

基于 Prometheus 构建 AI 全栈监控大盘，包括模型性能分析、Token 成本分析、GPU 资源异动分析等。

模型调用全链路诊断

基于 OpenTelemetry Trace 实现用户终端、网关、模型应用、模型服务、外部依赖工具等全链路追踪。

模型生成结果评估

构建统一日志分析平台，对模型调用日志进行二次评估分析，实现质量、安全、意图提取等语义检测。

AI 应用可观测体系



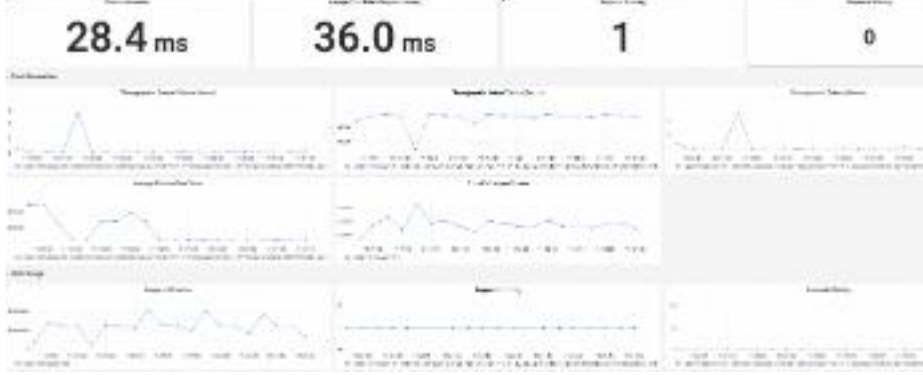
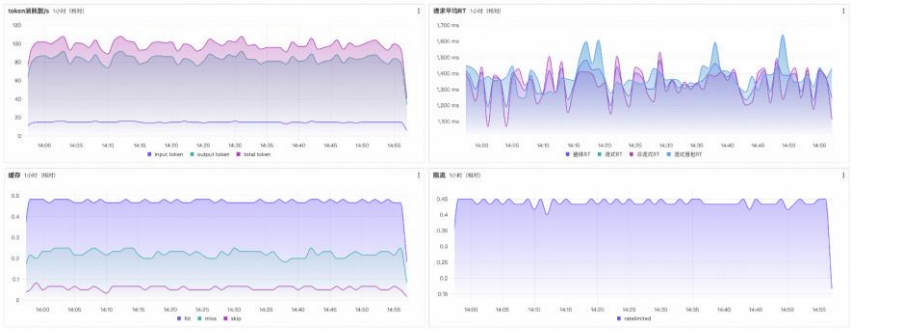
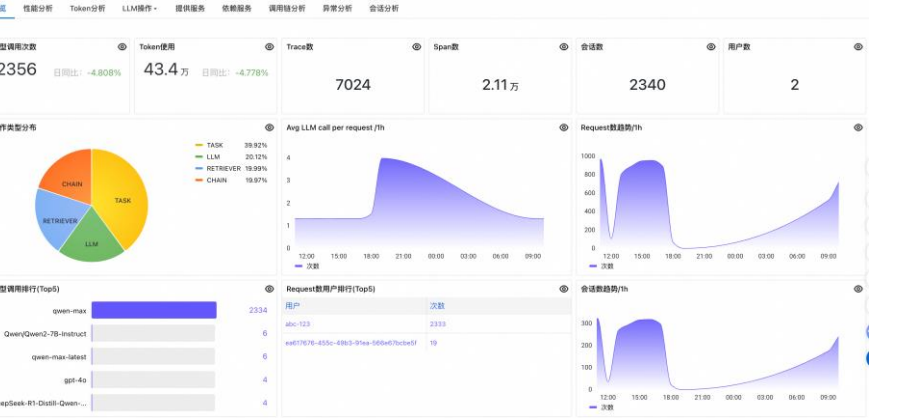
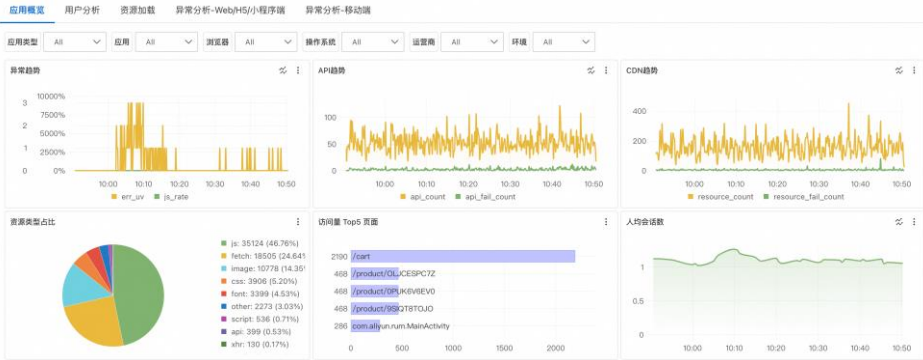
用户体验
监控

模型应用
监控

AI 网关
监控

模型服务
监控

基础设施
监控

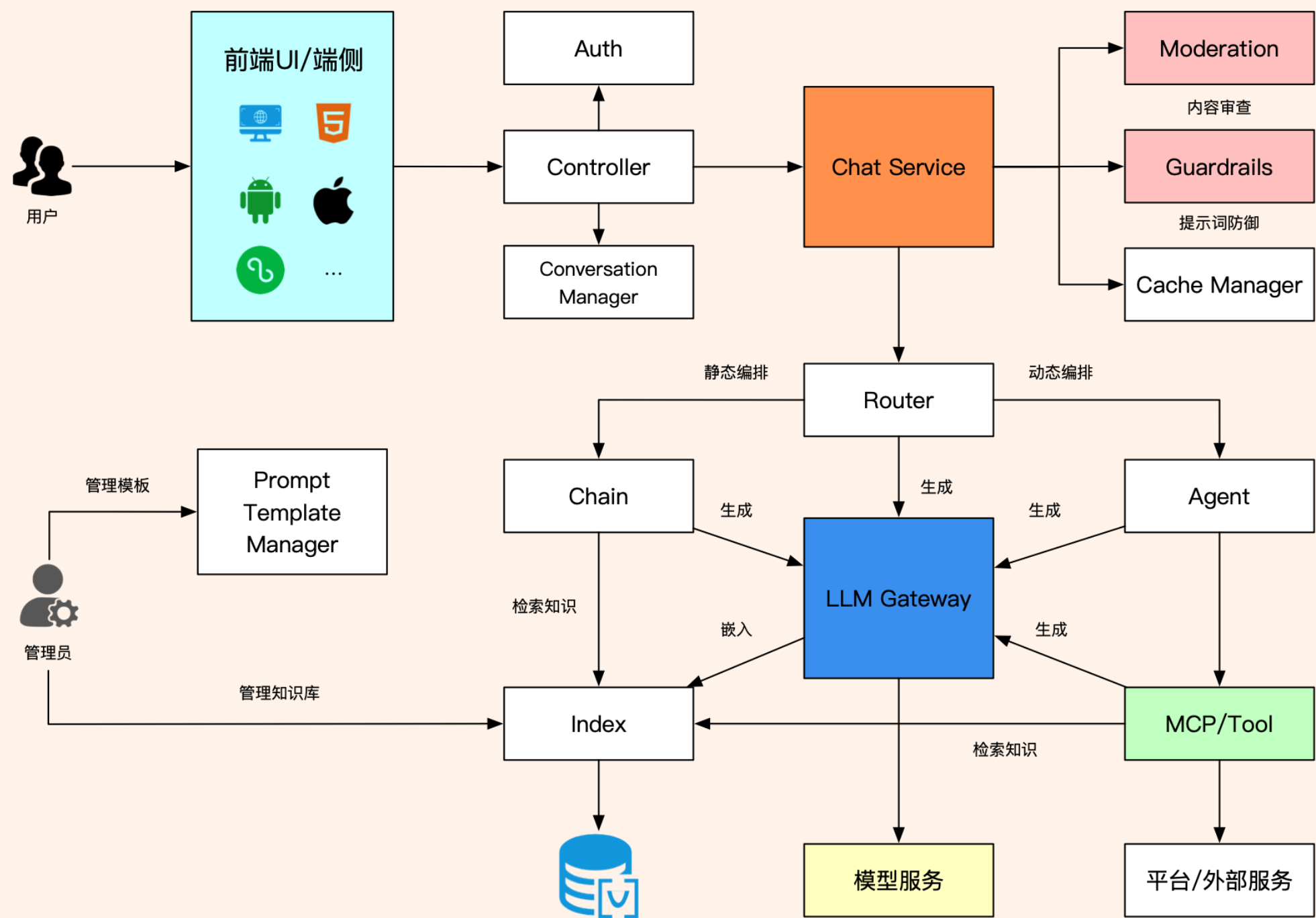


Tracing : 模型调用全链路诊断



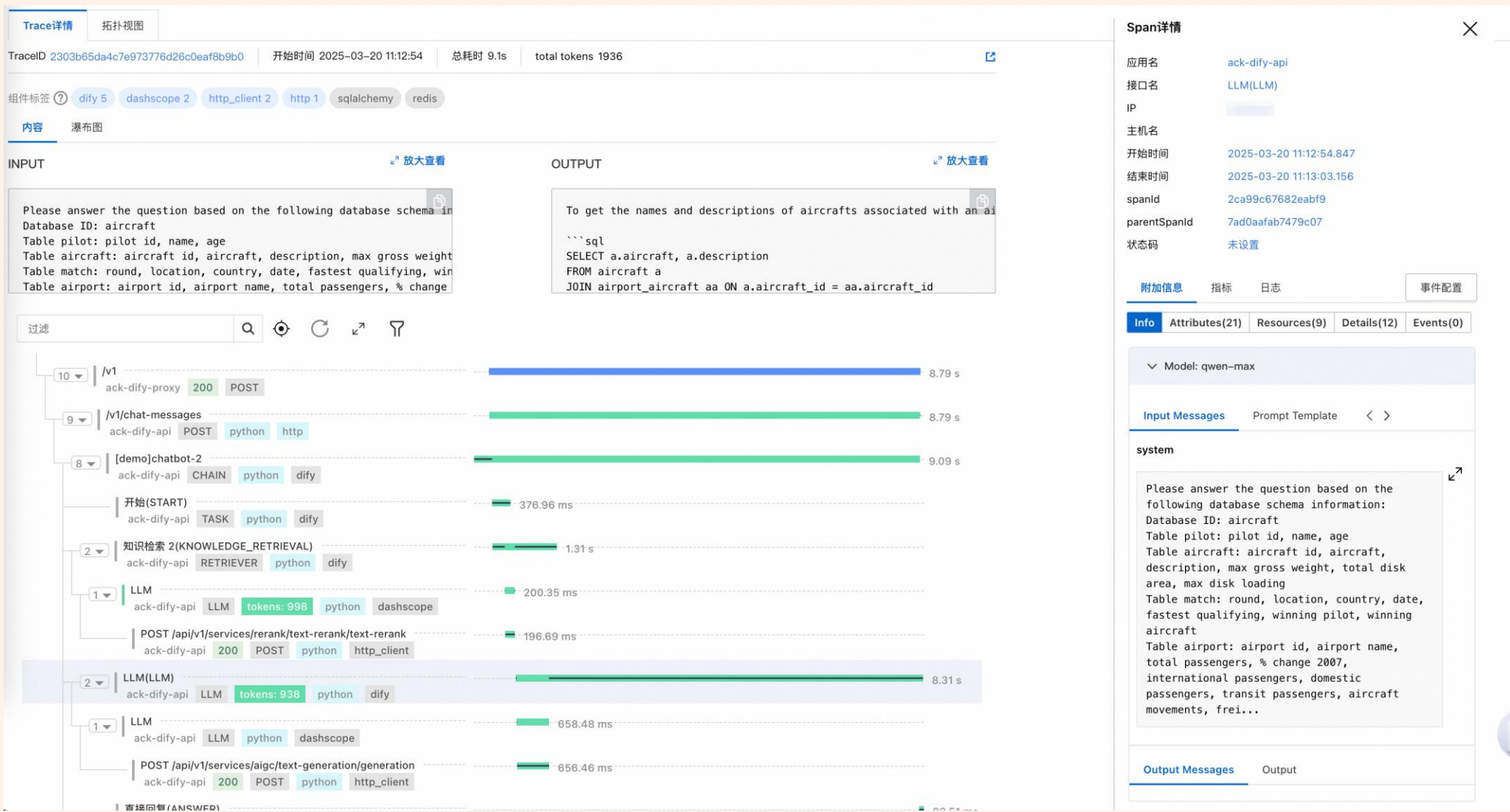
一个典型的 LLM ChatBot 应用架构

LLM ChatBot 包含前端 UI、对话服务、模型应用、模型服务、AI网关、向量数据库、外部Tool等众多组件。



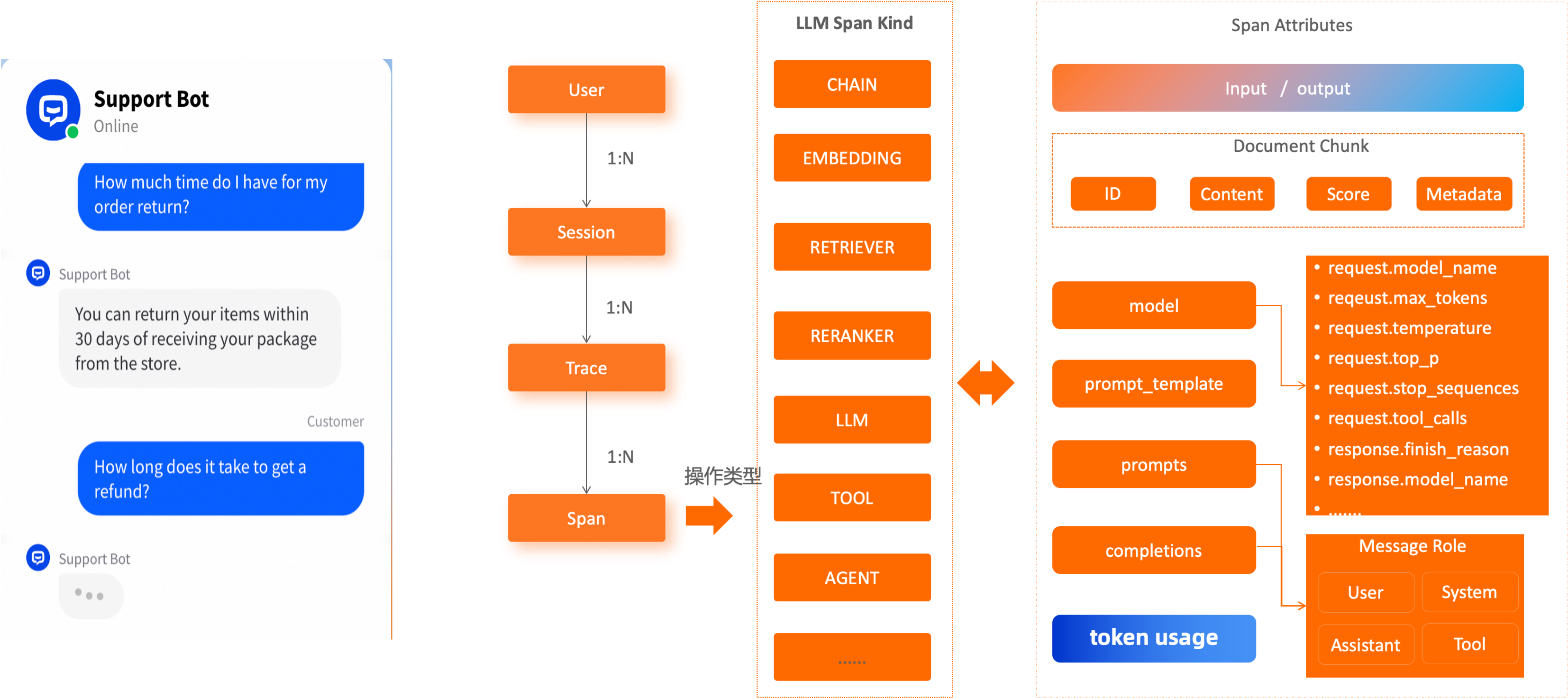
LLM Trace 端到端全链路追踪

基于 OpenTelemetry W3C 协议实现 LLM Trace 端到端全链路追踪。详细记录每一次模型调用过程，包括 Prompt、Input/Output、Token、TTFT (Time to first Token) 等。



面向 LLM 应用的领域化 Trace 语义

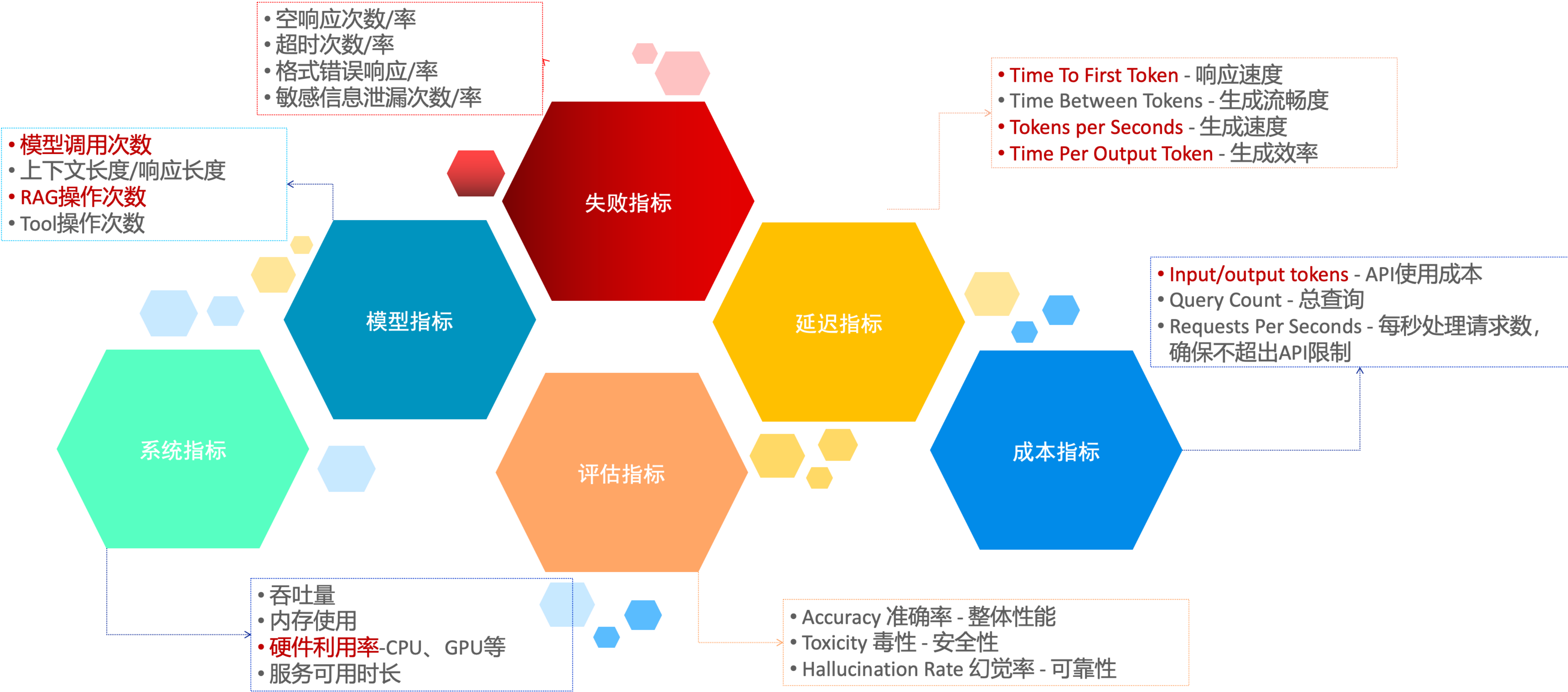
以会话串联用户交互问答过程，以 Trace 承载应用全链路交互节点，定义领域化的操作语义、标准化存储以及可视化关键内容。



LLM 应用可观测需要关注哪些指标?



基于 Prometheus + Grafana 实现 AI 全栈指标统一存储与可视化，构建 LLM 领域监控大盘，如模型性能分析、Token 成本分析等。



基于 OpenTelemetry 的高质量数据采集

以 OpenTelemetry Python Agent 为底座，增强大模型领域语义规范与数据采集，提供多种性能诊断数据，全方位自监控保障稳定高可用。



拥抱开源，面向大模型应用量身打造



一键安装接入

K8S环境支持修改YAML实现简单自动接入，initContainer机制注入探针，支持批量规模化自动接入



埋点更精细

针对框架埋点丰富优化，提供丰富的指标，支持用户添加自定义属性信息并在整条trace中透传

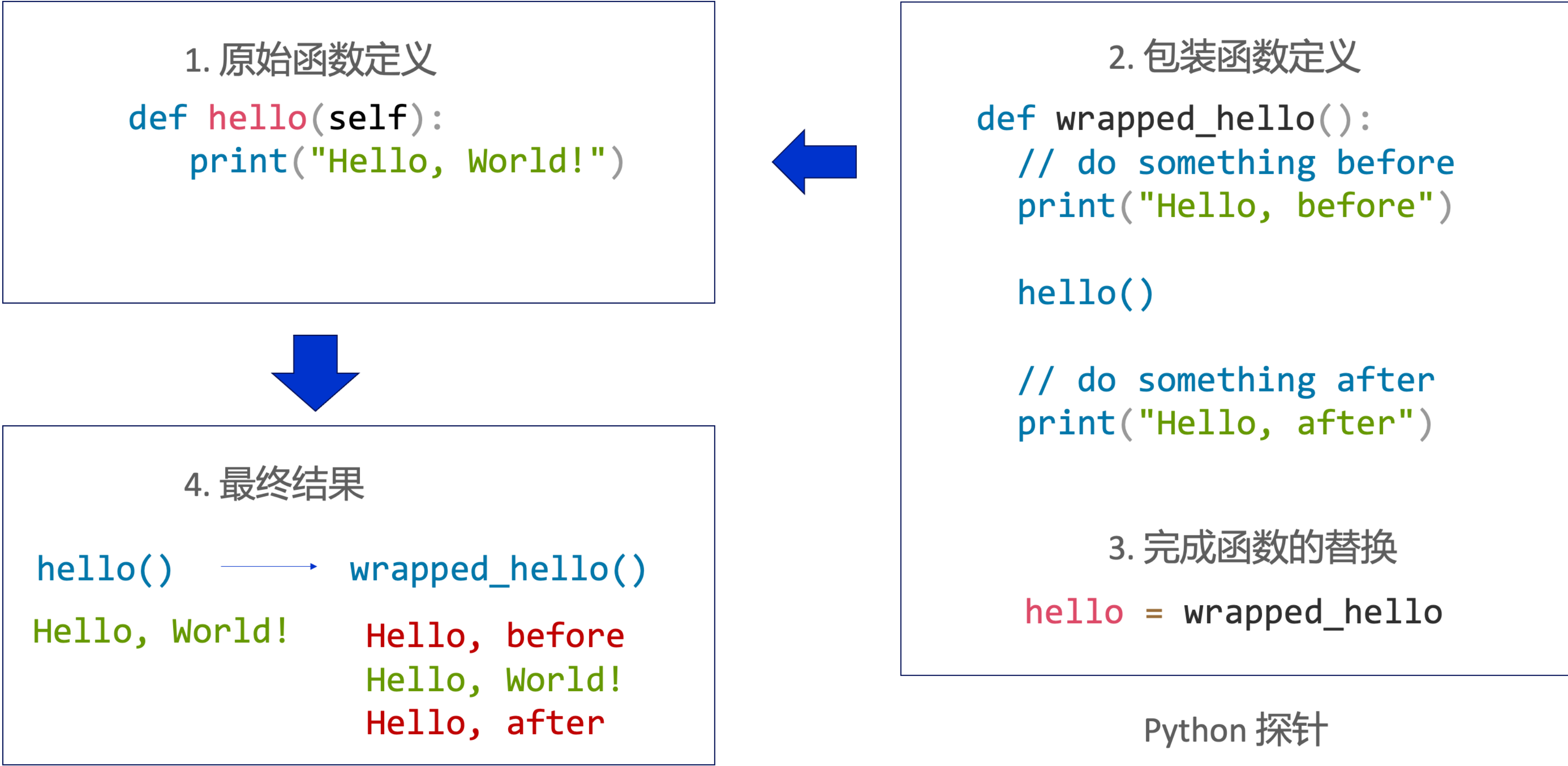


性能稳定性

埋点上报开销优化，降低应用性能影响，通过管控策略可以进一步降低开销在可控范围

Python 探针无侵入埋点的实现原理

Python 语言提供一种装饰器模式，底层采用了一种 monkey patch 的机制，允许程序在运行时改变一个模块或者类的属性和方法，实现无侵入的可观测代码插桩逻辑。



Python 探针无侵入埋点的实现原理



对比项	自研 Python 探针	开源 OTel Python 探针	OpenInference LLM 探针
大模型插件支持	全面支持，包括 OpenAI/Langchain/LlamaIndex/Promptflow/Dify/通义千问/vLLM/SGLang	不支持，OpenAI 尚在开发中	部分支持，OpenAI/Langchain/LlamaIndex
大模型埋点丰富度	埋点丰富，支持采集TTFT首包耗时、输入输出、Token消耗、模型名称、RAG过程等。支持异步流式场景采集	不支持	支持输入输出、Token消耗、模型名称、RAG过程等 不支持大模型指标
基础插件支持	部分支持，常用HTTP插件均支持	完整支持，HTTP/NoSQL/Database	不支持
OTel Gen AI 语义	基本遵循，部分有差异	遵循最新的 GenAI 规范	基本遵循
协程支持 (gevent)	完整支持	不支持	不支持
多进程支持 (unicorn/gunicorn)	完整支持	部分支持，会启动失败	不支持
Python 基础监控	支持 CPU/内存/GC 次数等指标	不支持	不支持
性能	2000 QPS CPU 占用 < 10%	2000 QPS CPU 占用 15%	/
稳定	支持收敛能力	不支持	不支持

面向流式场景的 LLM Span 分段采集与合并

随着 SSE 协议在大模型推理场景的广泛应用，针对流式数据采用客户端分段采集+服务端合并还原，以平衡客户端侧性能与数据分析易用性。

- 1

拆分长文本长耗时 LLM Span

每个分片作为一个Event单独上报
- 2

模型调用结束，上报主Span

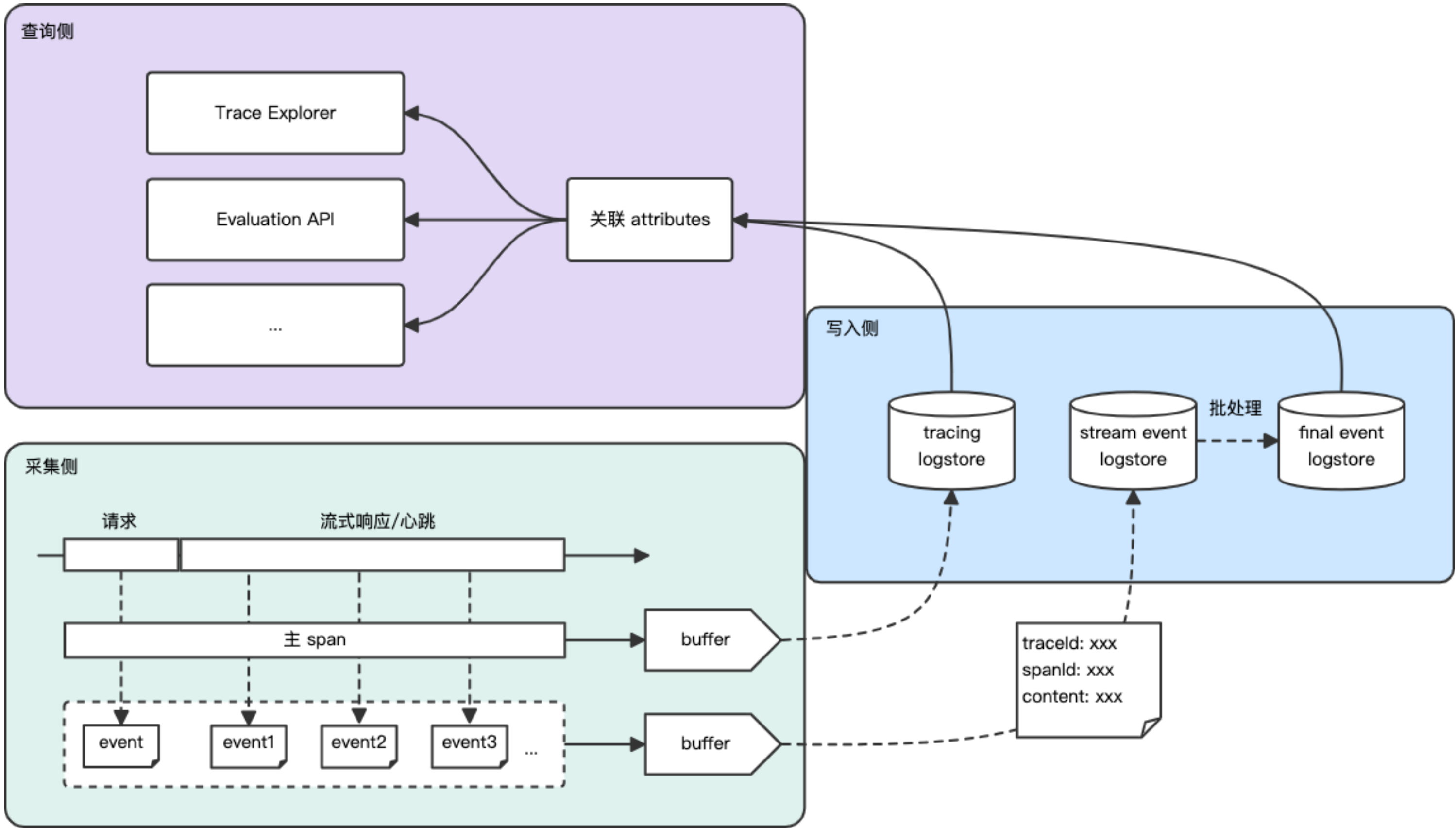
记录分片数量、TTFT、Token等信息
- 3

合并 Events 还原上下文

当次调用的 Events 合并为单行数据
- 4

模型调用内容二次评估分析

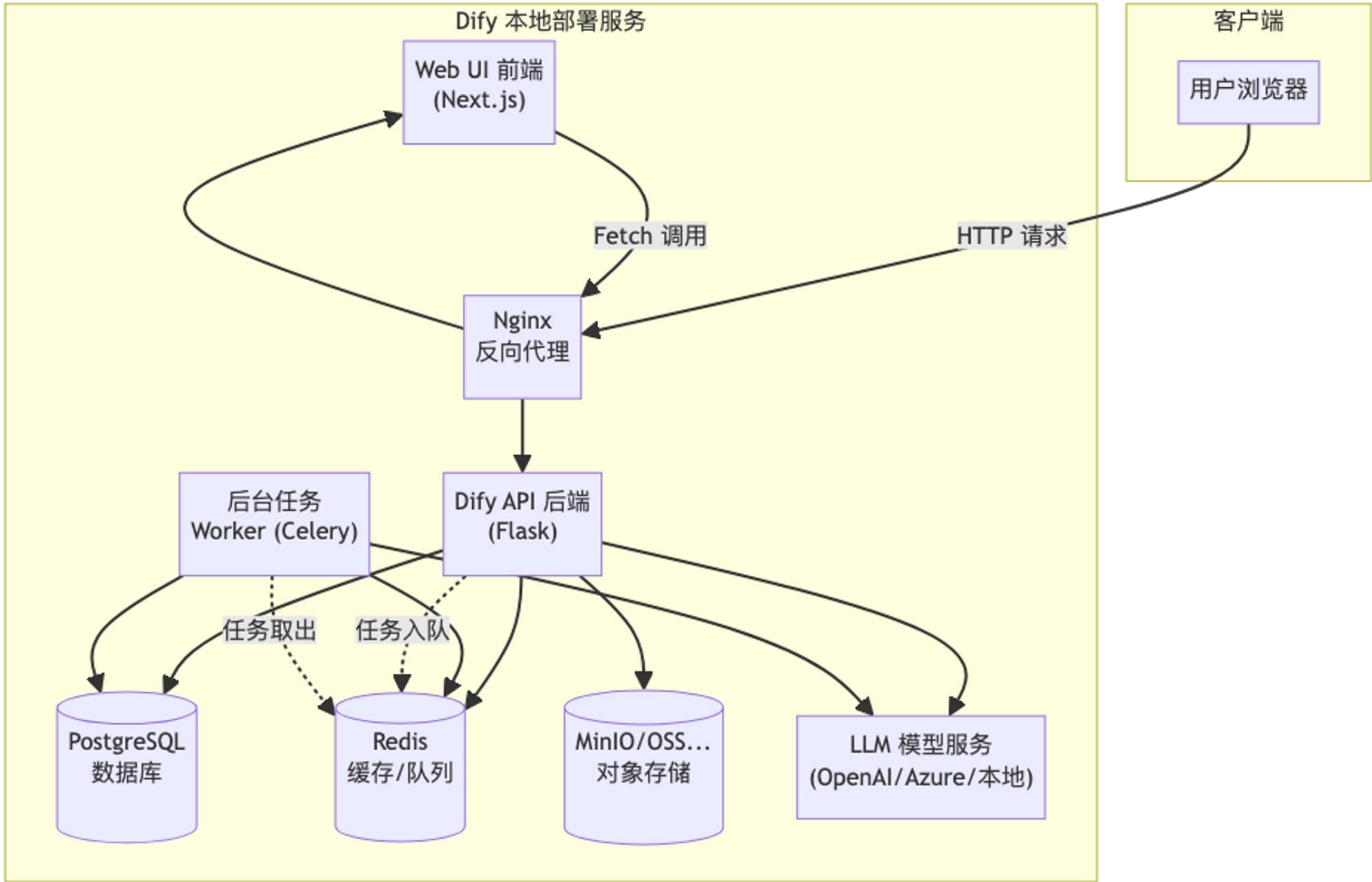
下游评估任务/分析模块进行查询



Dify 可观测最佳实践

Dify 生产实践建议：

- 1. Nginx 调大 NGINX_CLIENT_MAX_BODY_SIZE
- 2. PGSQL 数据库连接池大小：建议增加到 300 或以上
- 3. Redis 使用哨兵模式提升高可用性
- 4. 使用第三方向量数据库
- 5. 使用外置可观测组件
- 6. 存储配置建议使用云存储（如 OSS）

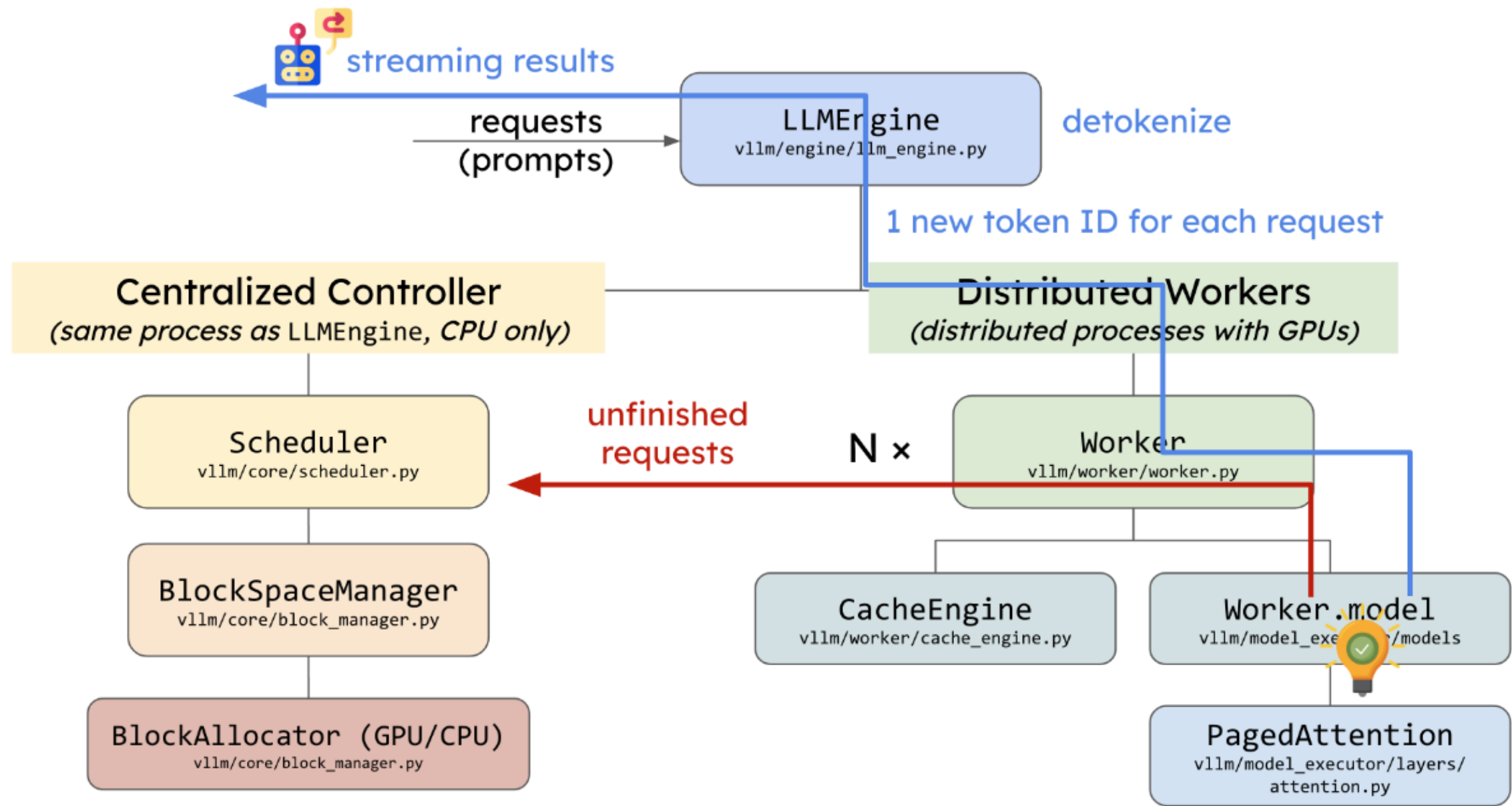


对比项	Dify 原生可观测	OTel 原生可观测	阿里云自研 Python 探针
接入方式	按应用接入，需为每个应用单独配置（如100个应用需接入100次），成本高。	需修改代码（侵入式），每次版本更新需调整；无侵入接入可能因gevent导致主进程挂起导致卡死，成本高。	一次接入，支持无侵入式集成， 接入成本低 。
观测维度	单一，仅观测 Dify LLM 应用层（如Workflow、Agent等）。	单一，仅观测平台服务层，缺少对 Dify LLM 应用层的观测。	多维度，覆盖： - 应用层（LLM组件） - 平台服务层（HTTP、数据库等） - 用户体验层 - 基础架构层（插件守护进程、沙箱、运行时等）。
分析维度	单应用孤立分析，仅能查看单应用的Token消耗等，无法全局分析。	无法分析用户层信息。	全面分析： - 全局 Token 统计 - 单应用/多应用层性能 - 平台服务层性能等。
观测链路	孤岛式链路追踪，仅支持单应用内部链路，无法跨应用或跨平台。	仅包含平台层链路，无法追踪具体应用或跨平台链路。	端到端全链路： Client → 网关(Nginx) → Dify Web → LLM应用 → 插件 → AI网关 → 模型(vLLM/SGLang)。
性能	一般，需将链路数据存储在 Dify PGSQL 数据库后再上报，查询效率较低。	较差，大数据量时易发生内存泄漏，内存未及时释放。	性能较好 ，数据旁路上报，开销小。
稳定性&SLA	不保证 SLA，但作为后处理流程不会导致服务崩溃。	不保证 SLA，存在崩溃风险（多线程与gevent协程锁竞争问题）。	保证SLA

vLLM/SGLang 推理性能可观测实战



vLLM 是一个高效的推理引擎，通过动态批处理、内存分块管理、KV缓存复用等方式大幅度提升了推理效率。



问题现象

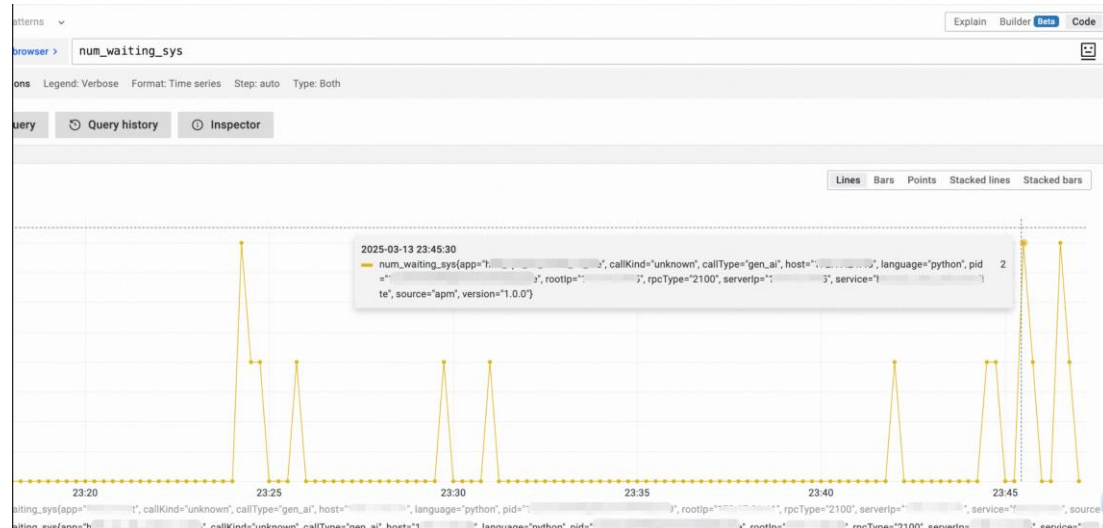
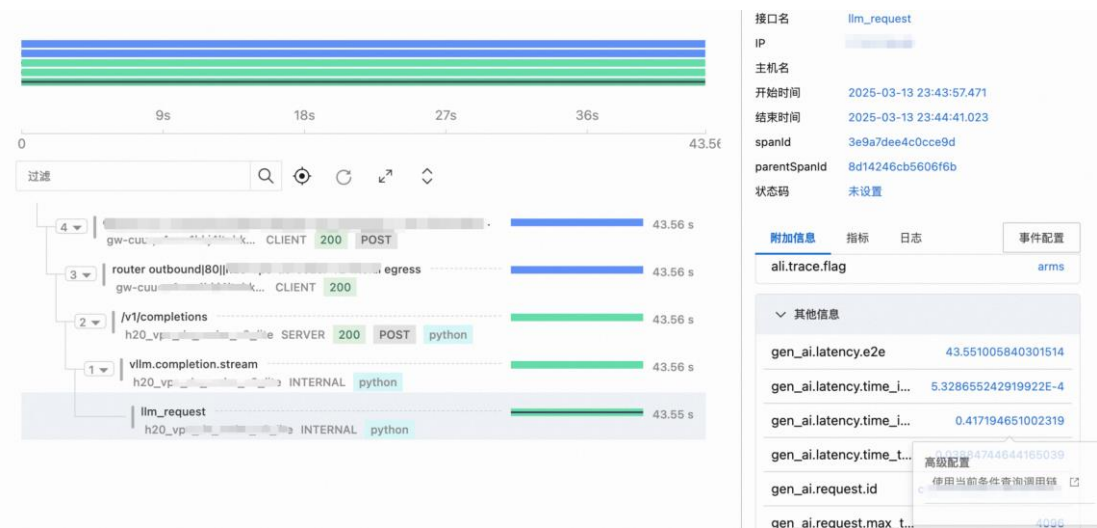
某业务通过 AI 网关调用自建 DeepSeek 模型服务发现请求耗时很高

排查思路

1. 通过 RequestId 查询到关联的 Traceld，再通过端到端调用链分析发现模型推理阶段耗时非常高。
2. 通过调用链和指标关联，排查同一时刻相关指标是否异常
 1. 首先观察 TTFT 指标，确认正常，排除 prefill 阶段问题
 2. 检查 TPOT 指标，确认正常，排除 decode 阶段问题
 3. 检查推理引擎中正在执行的请求和排队的请求数量，发现正在执行的请求数量激增，同时出现较多的排队请求
 4. 最终确认是请求数量过多，超过当前处理能力后导致排队

建议

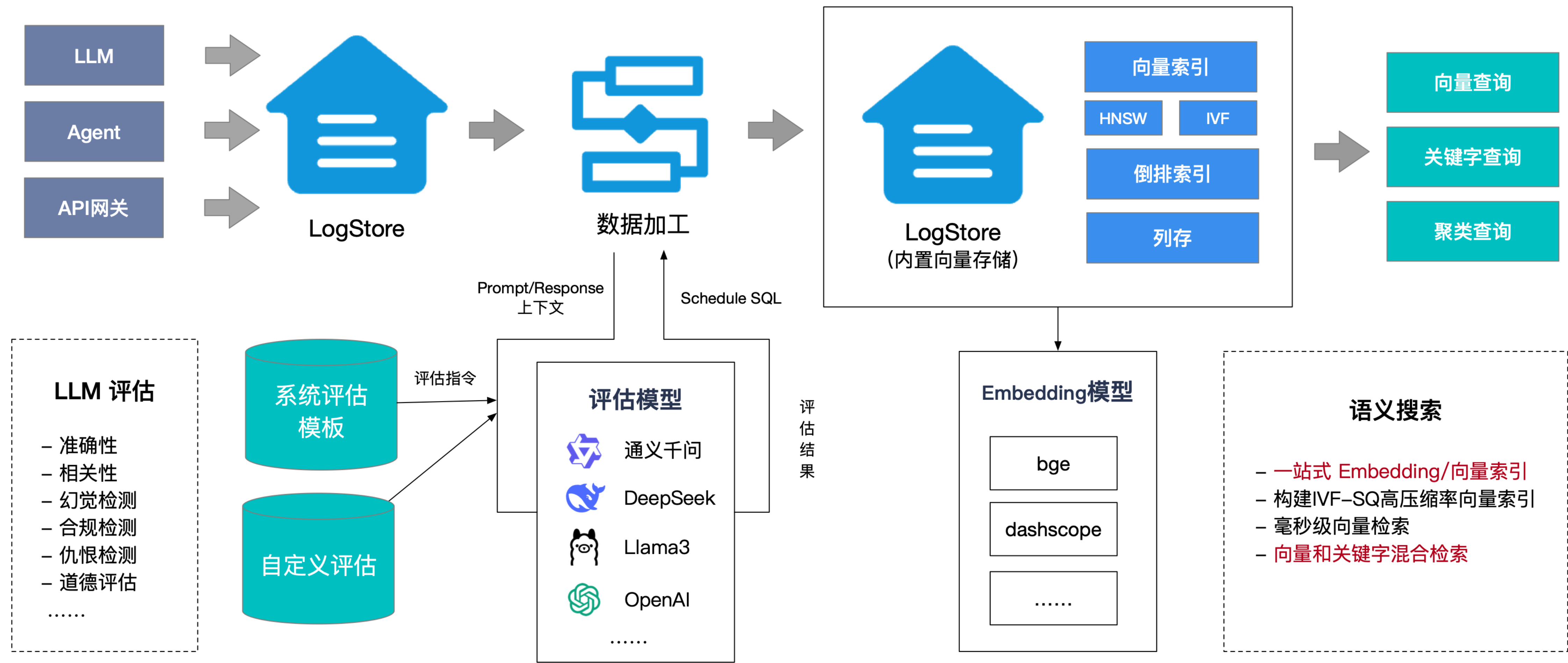
增大推理引擎中请求队列的大小配置



基于 LLM 实现模型生成结果自动化评估

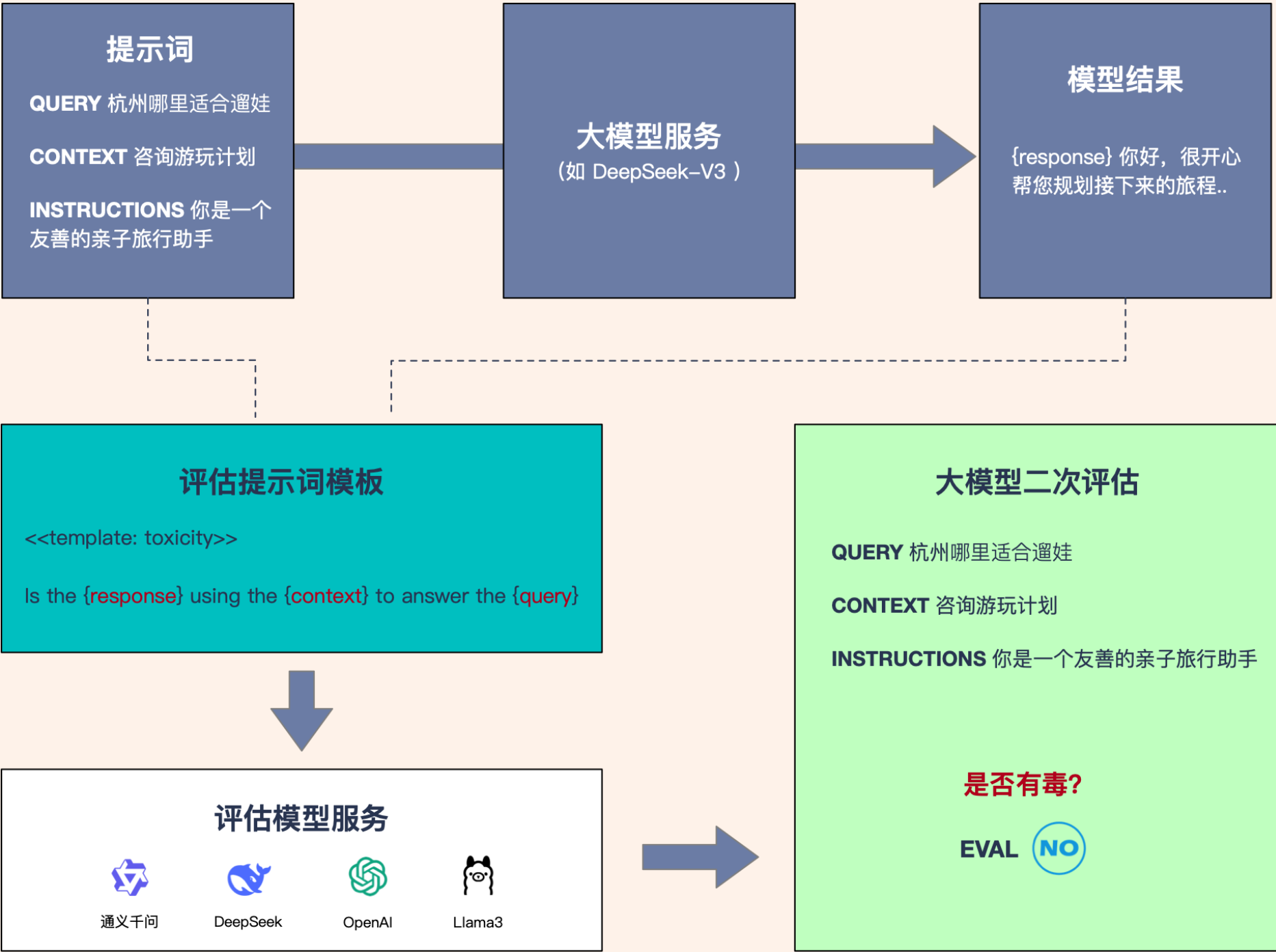


系统评估（内置10+模板）降低门槛、自定义评估深度提升生成效果。一站式Embedding/向量索引、向量和关键字混合检索简化开发流程。



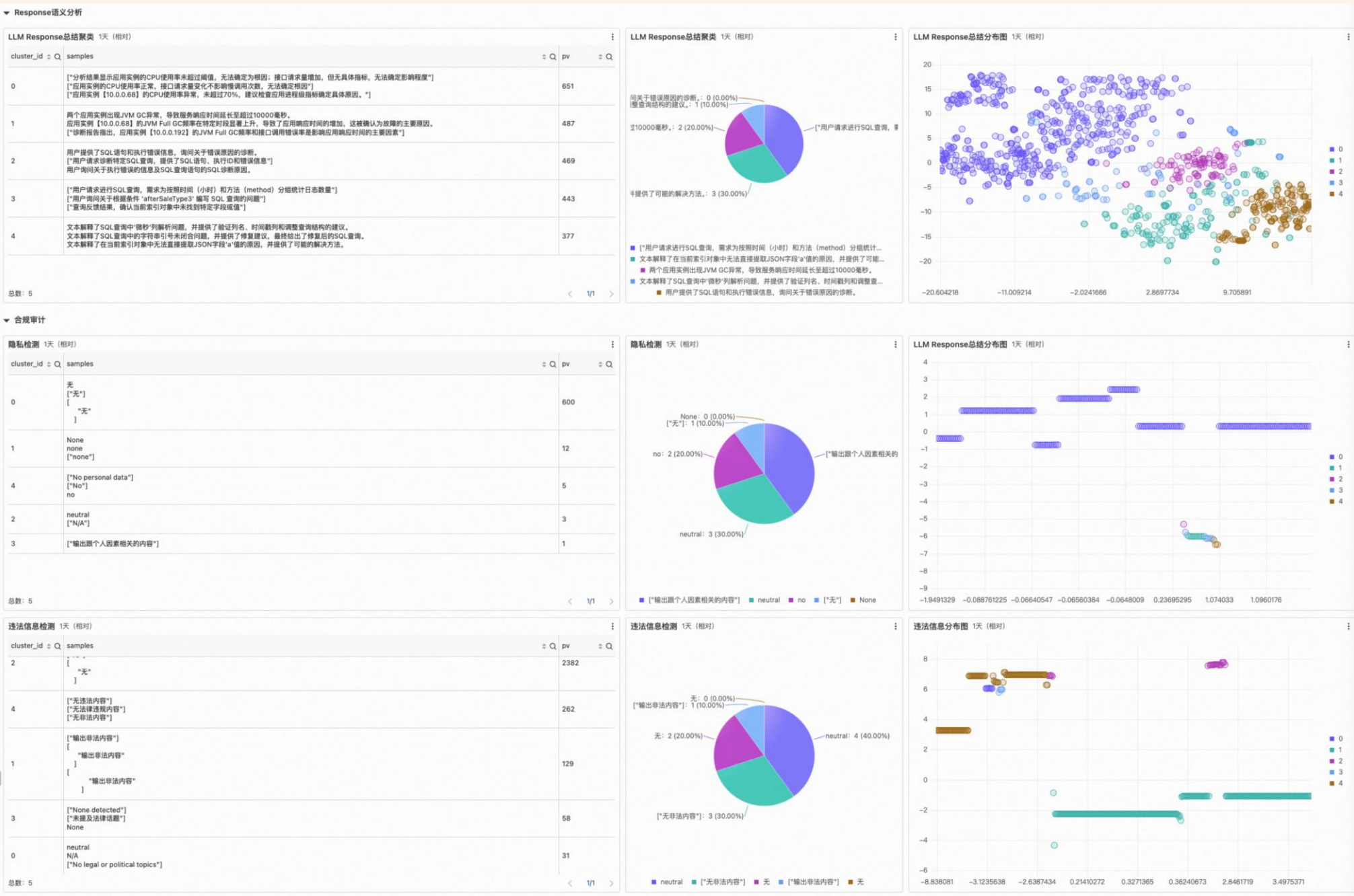
评估，为了更好的生成

为了更好的提升大模型生成效果，可以通过新的模型对原有模型 Prompt/Response 进行二次评估（质量、安全等）。



自动化评估（定量/定性）

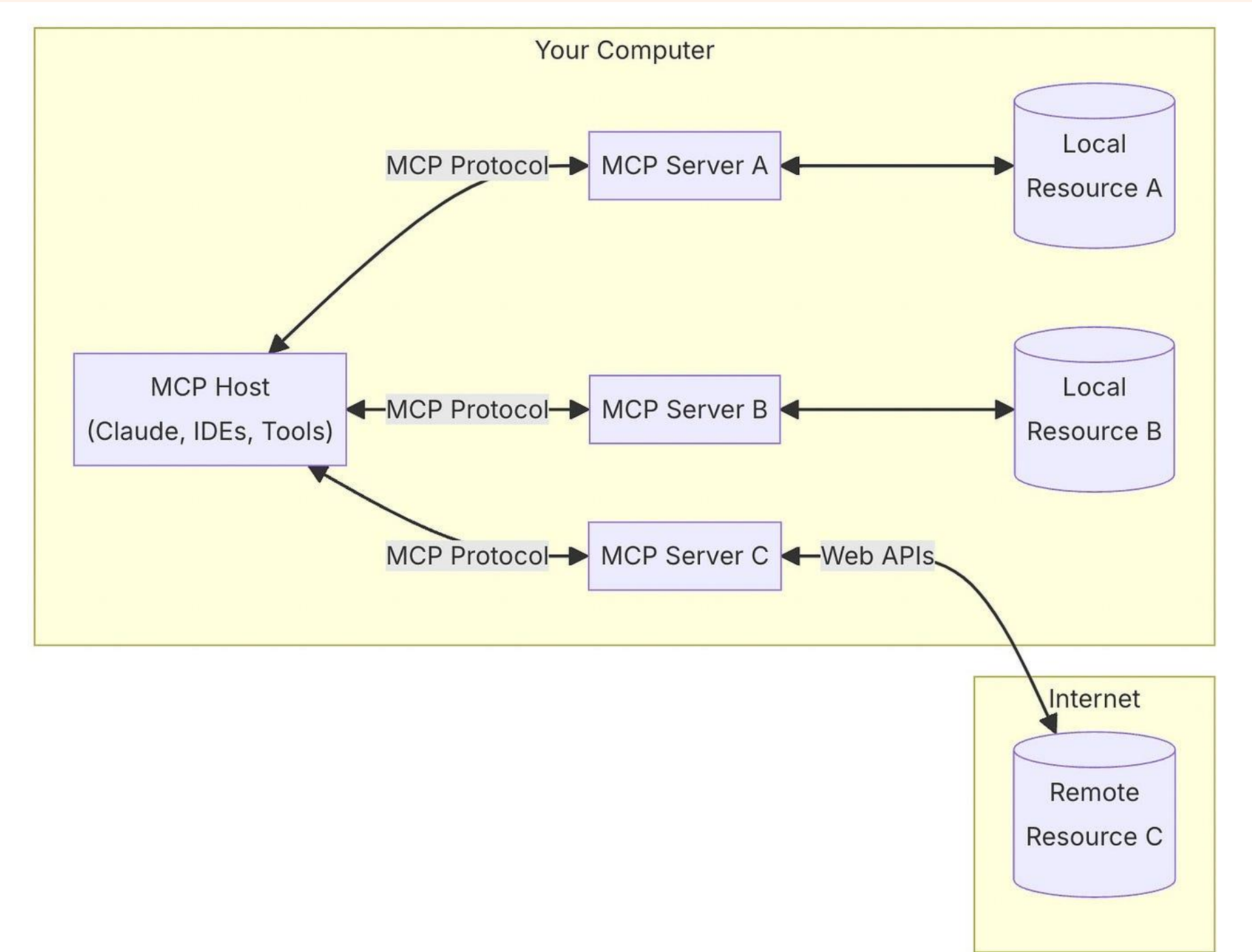
基于 Trace 或模型日志中记录的上下文内容，可以实现自动化定量/定性评估，比如合规性检查、用户意图提取等。



MCP 为大模型生态带来了哪些变化和挑战？

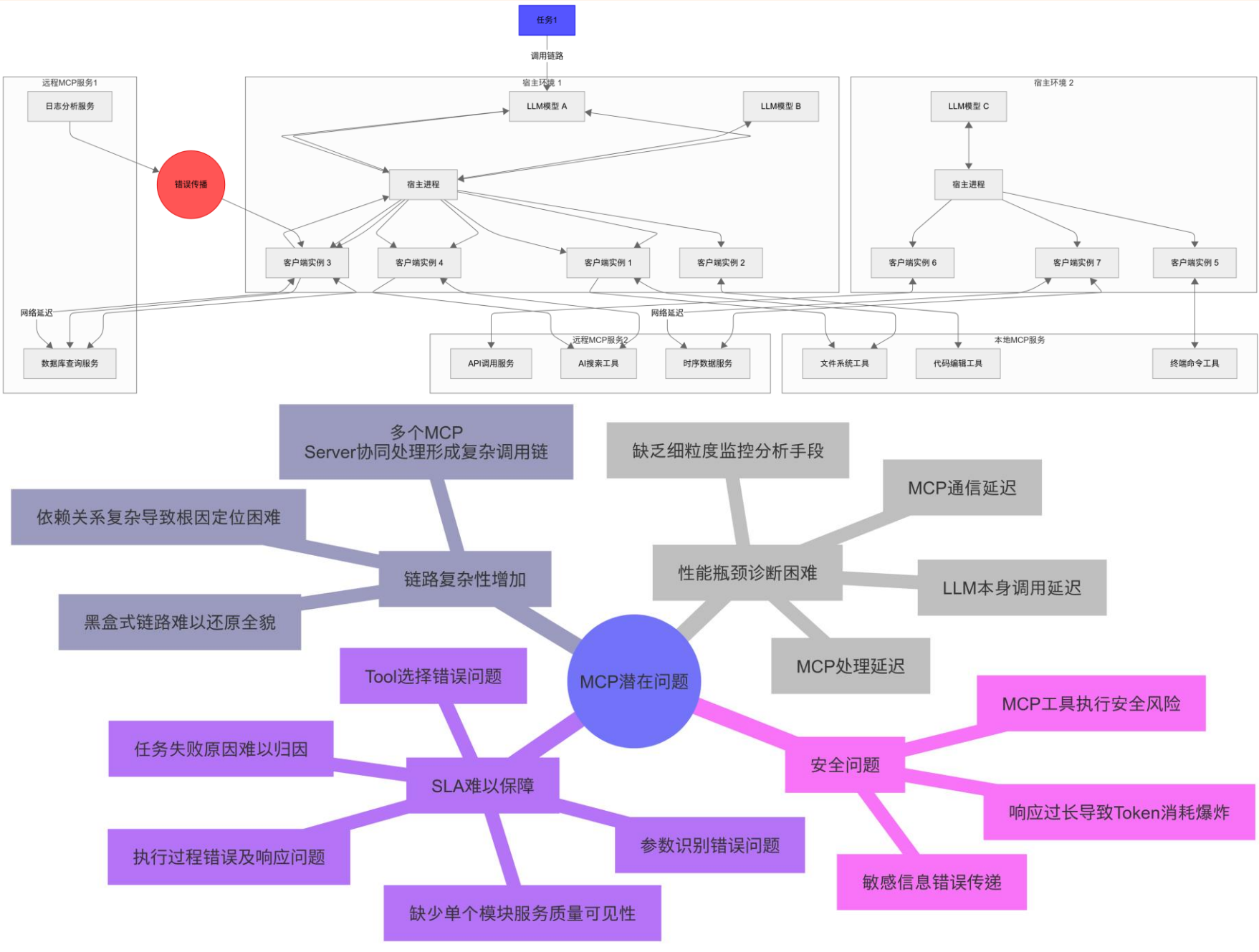
通用连接器解决 “N×M” 集成问题

MCP (Model Context Protocol) 为人工智能模型和开发环境之间建立统一的上下文交互提供了标准化方法。



模型&MCP多轮交互导致 “熵增”

LLM模型、宿主环境和各种 MCP 服务器之间的相互连接，会导致链路复杂性提升、性能瓶颈诊断困难、SLA难以保障以及安全等问题，可观测是降低 “熵增” 最有效的办法。

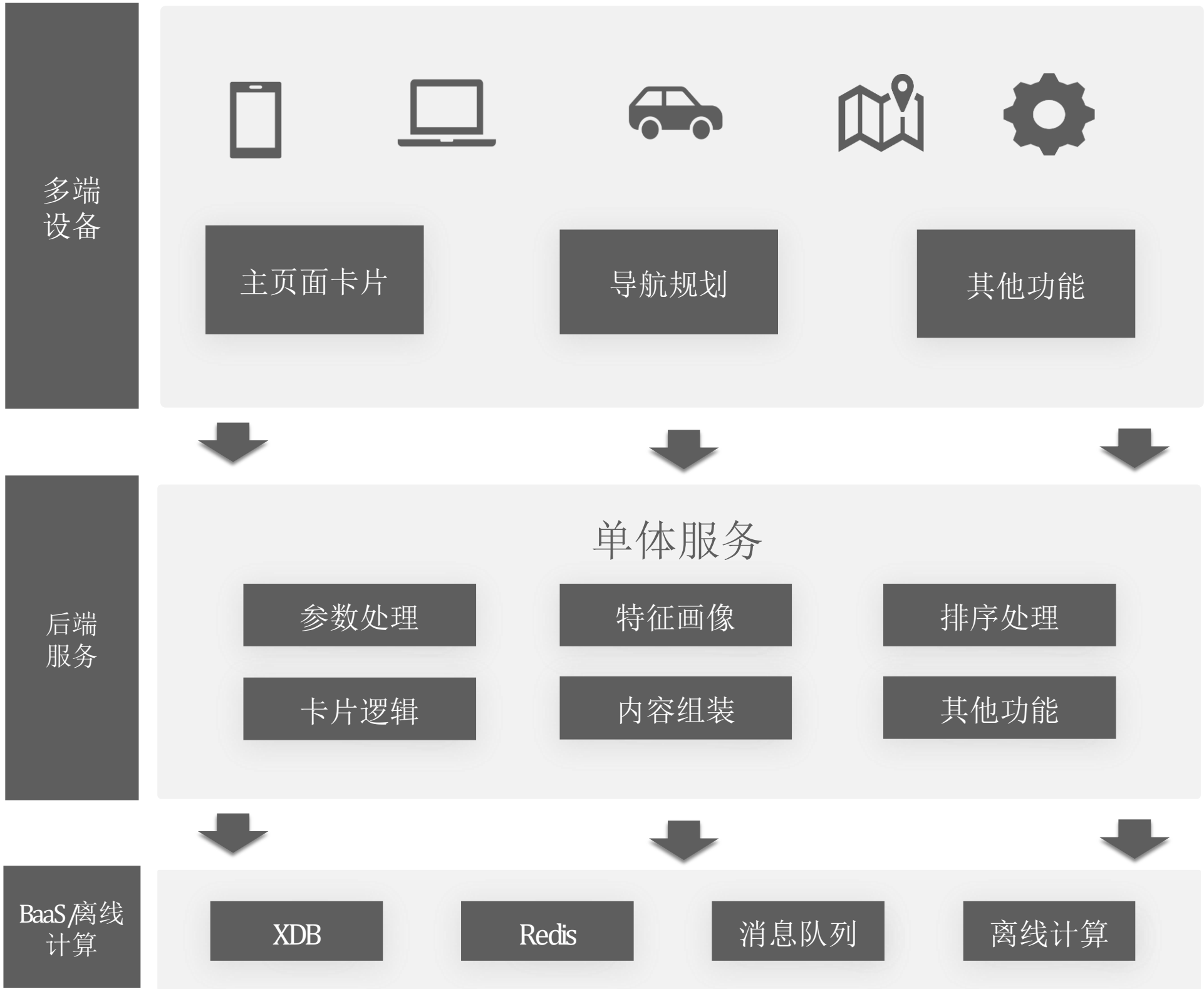


13 AI 应用开发新范式对企业的影响

高德业务投放平台 Serverless 实践 (API First架构)

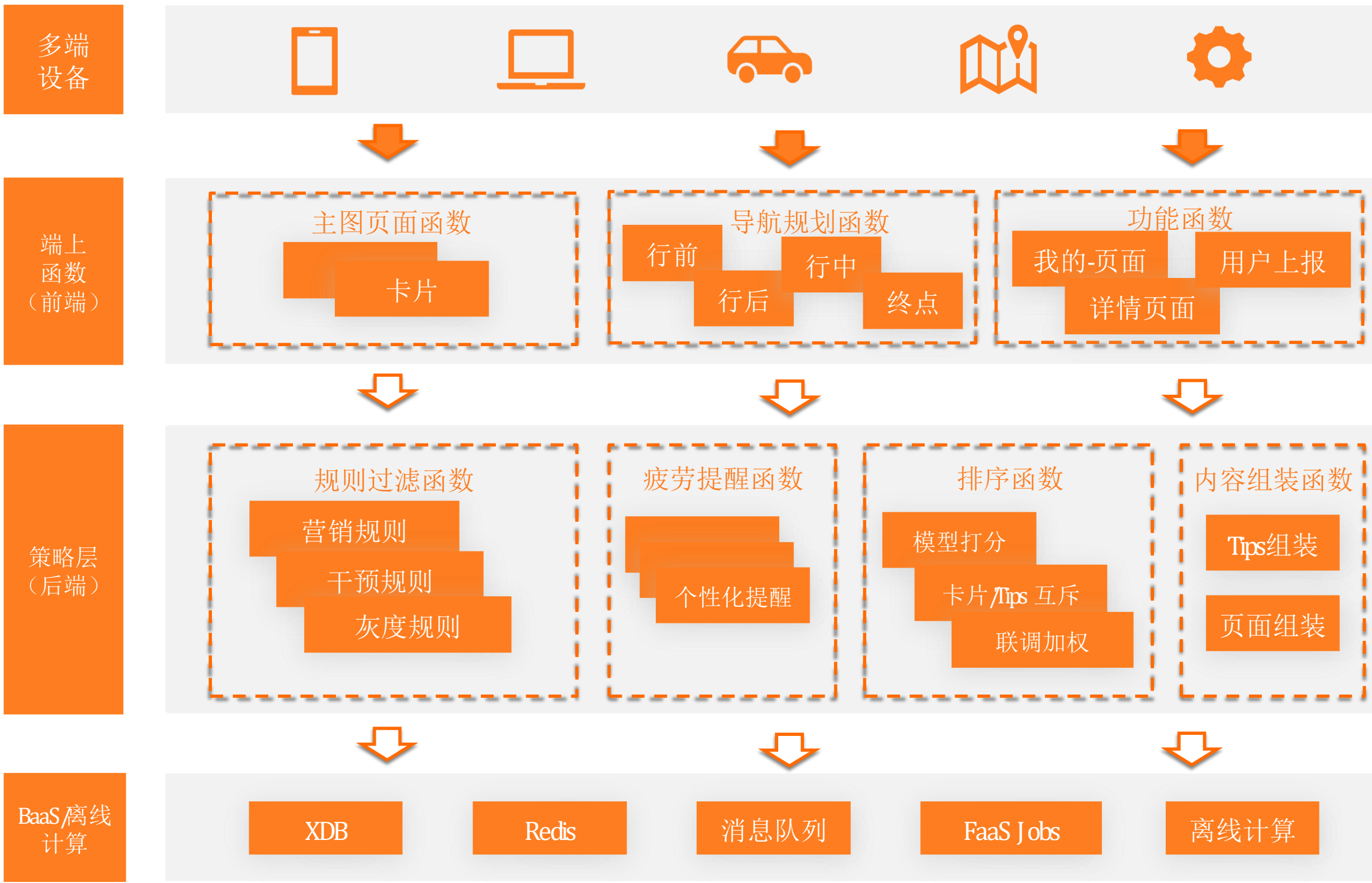


上一代架构



客户端太重 业务紧耦合 研发迭代慢 资源成本高

Serverless 架构

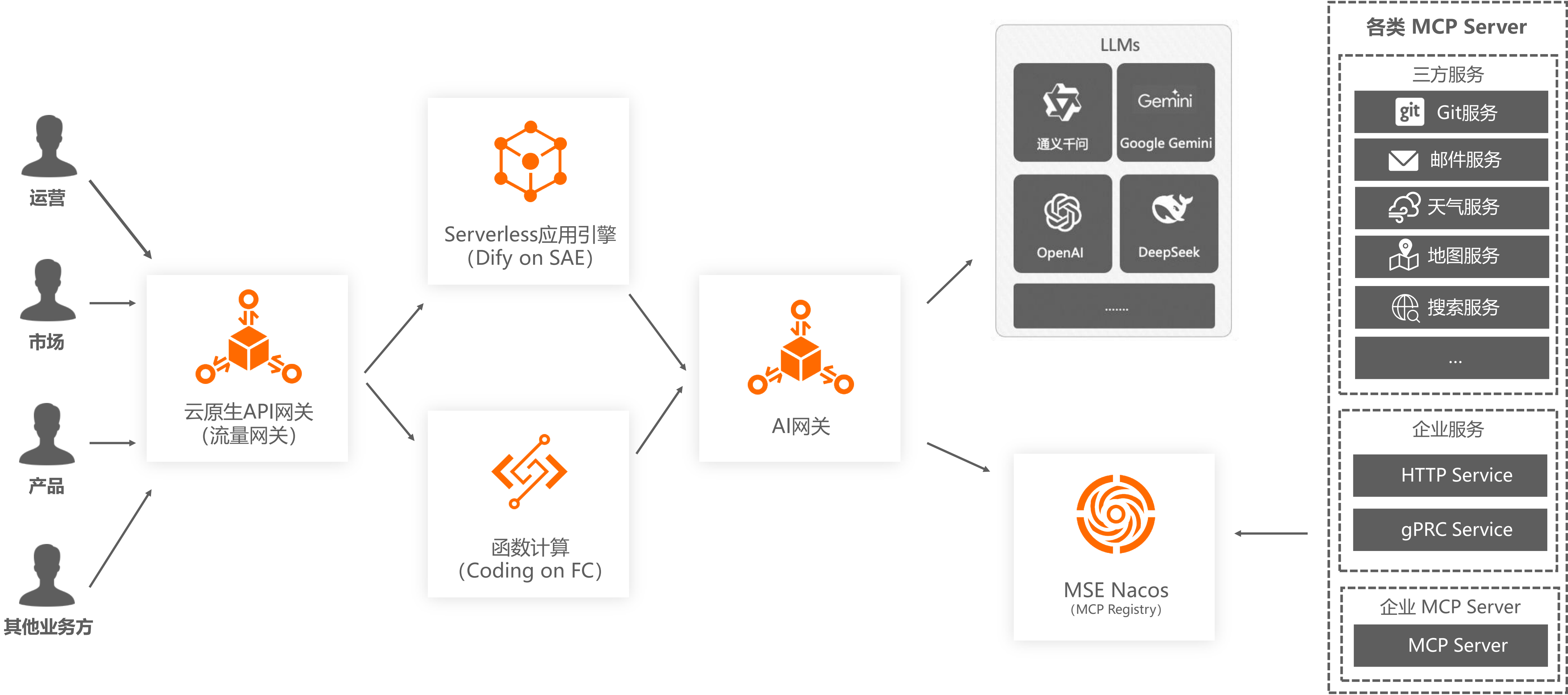


全链路 Serverless, 灵活弹性

按需快速组装业务功能

最小粒度灰度发布

MCP Server First



低代码模式，拖拖拽拽构建业务流程

业务流程里的节点都是以大白话描述业务需求

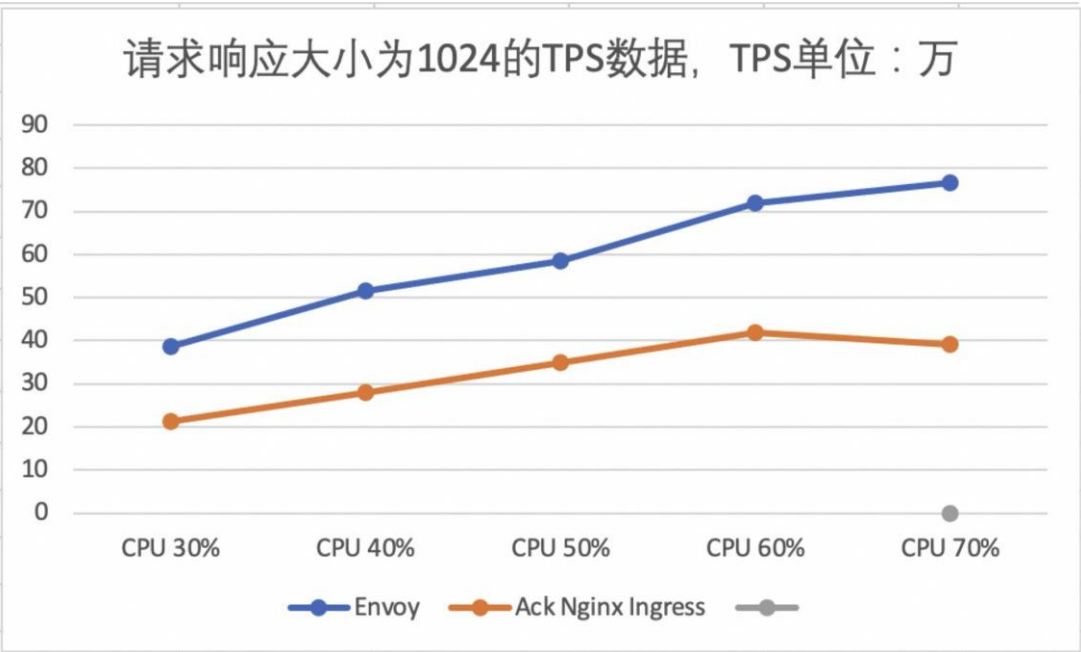
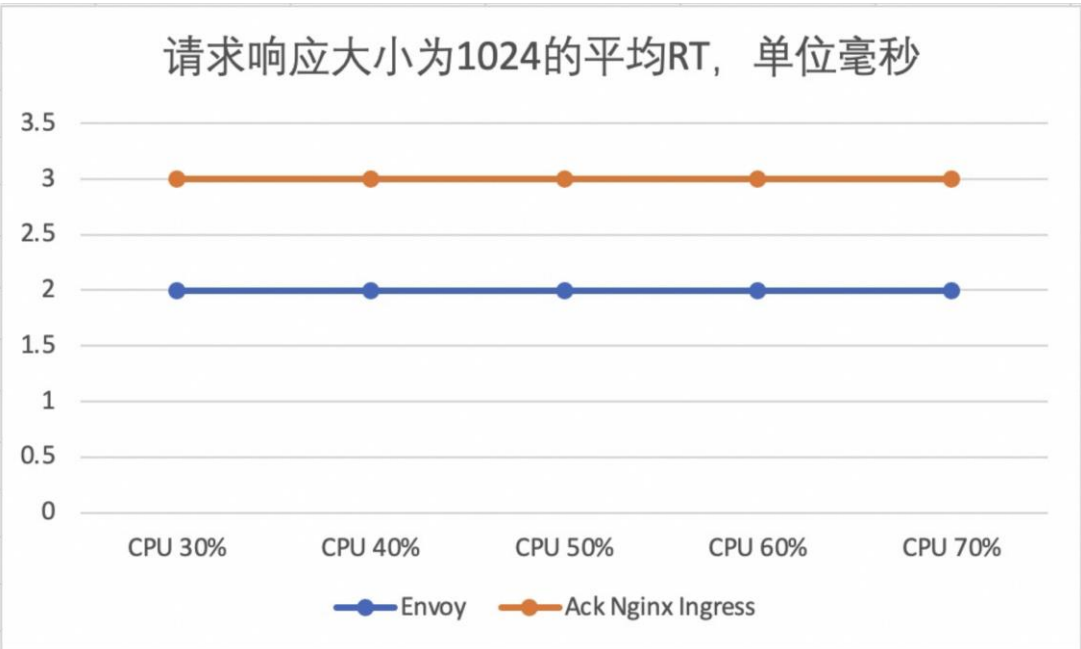
只要MCP Server足够丰富，描述调试足够准确
就不怕业务方攒不出他们想要的业务流程



附录 云原生API网关底座核心优势

云原生API网关 – 高性能（比自建性能高1-5倍）

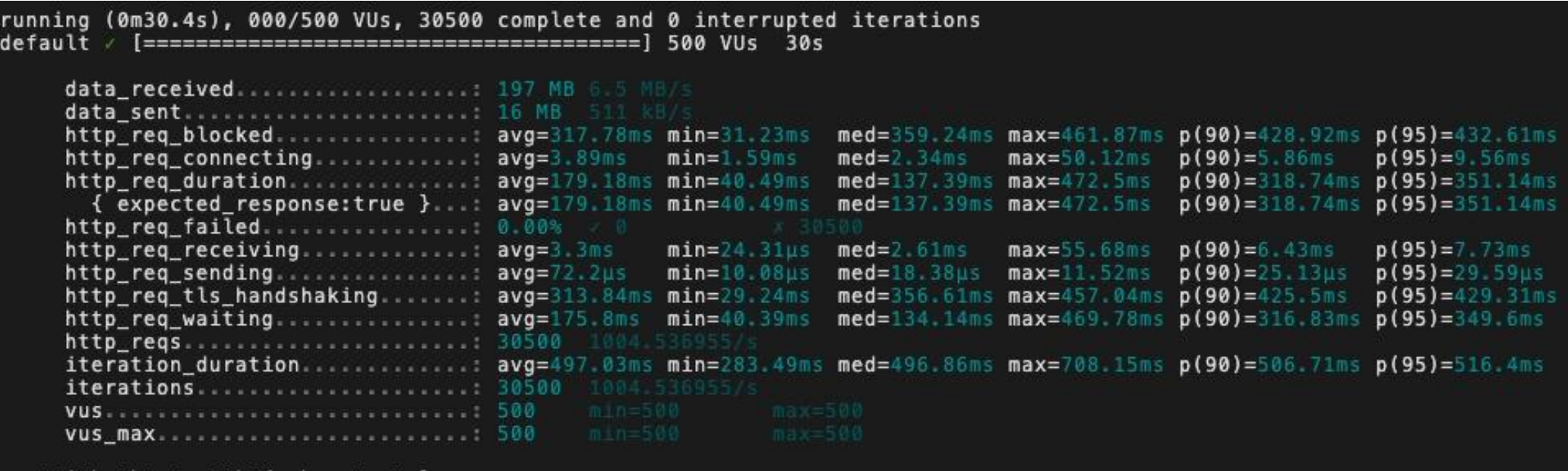
1、Nginx Ingress高出约 90%



网关规格：16C32G * 4 节点
ECS 型号：七代机 (ecs.c7.xlarge)

2、硬件加速HTTPS QPS 提升约112%， RT下降50%

加速前：



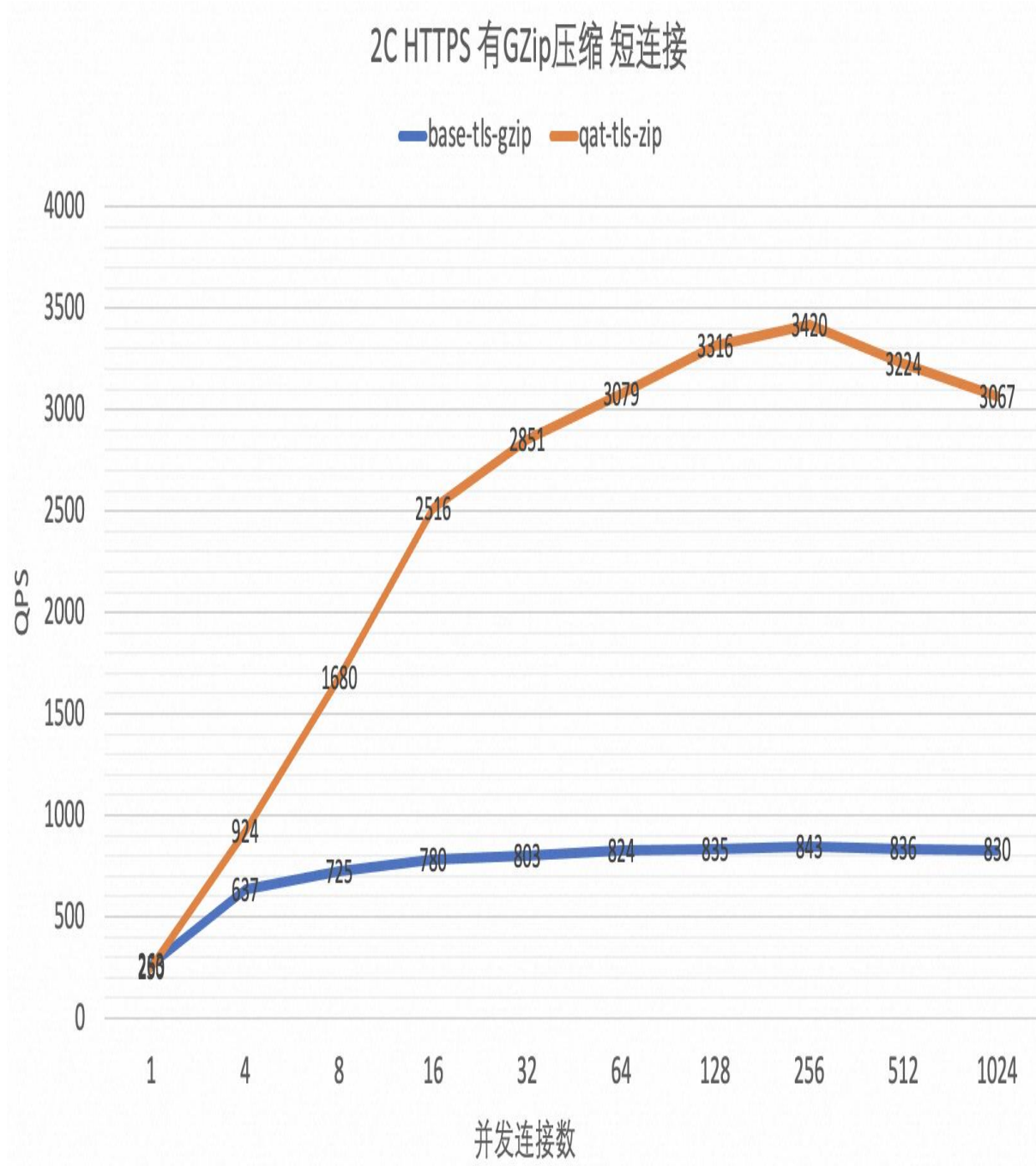
加速后：



注：测试采用HTTPS短连接且关闭session ticket复用。

网关规格：1 核 2 G * 1 节点

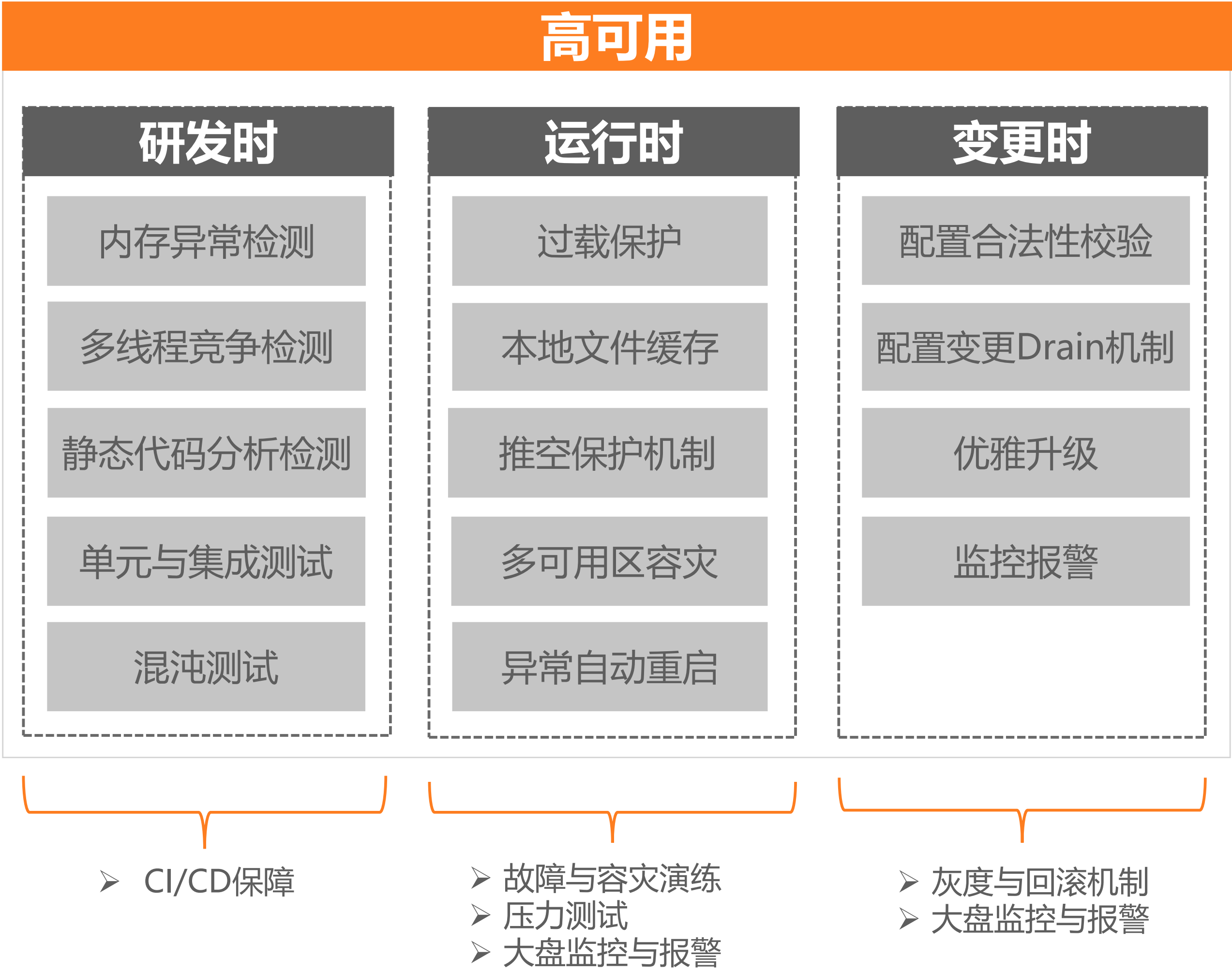
3、硬件加速压缩/解压缩提升 300%



网关规格：2C4G * 1 节点
ECS 型号：八代机

4、结合阿里大规模生产经验从操作系统/网络/内核深度调优，性能提升 40%

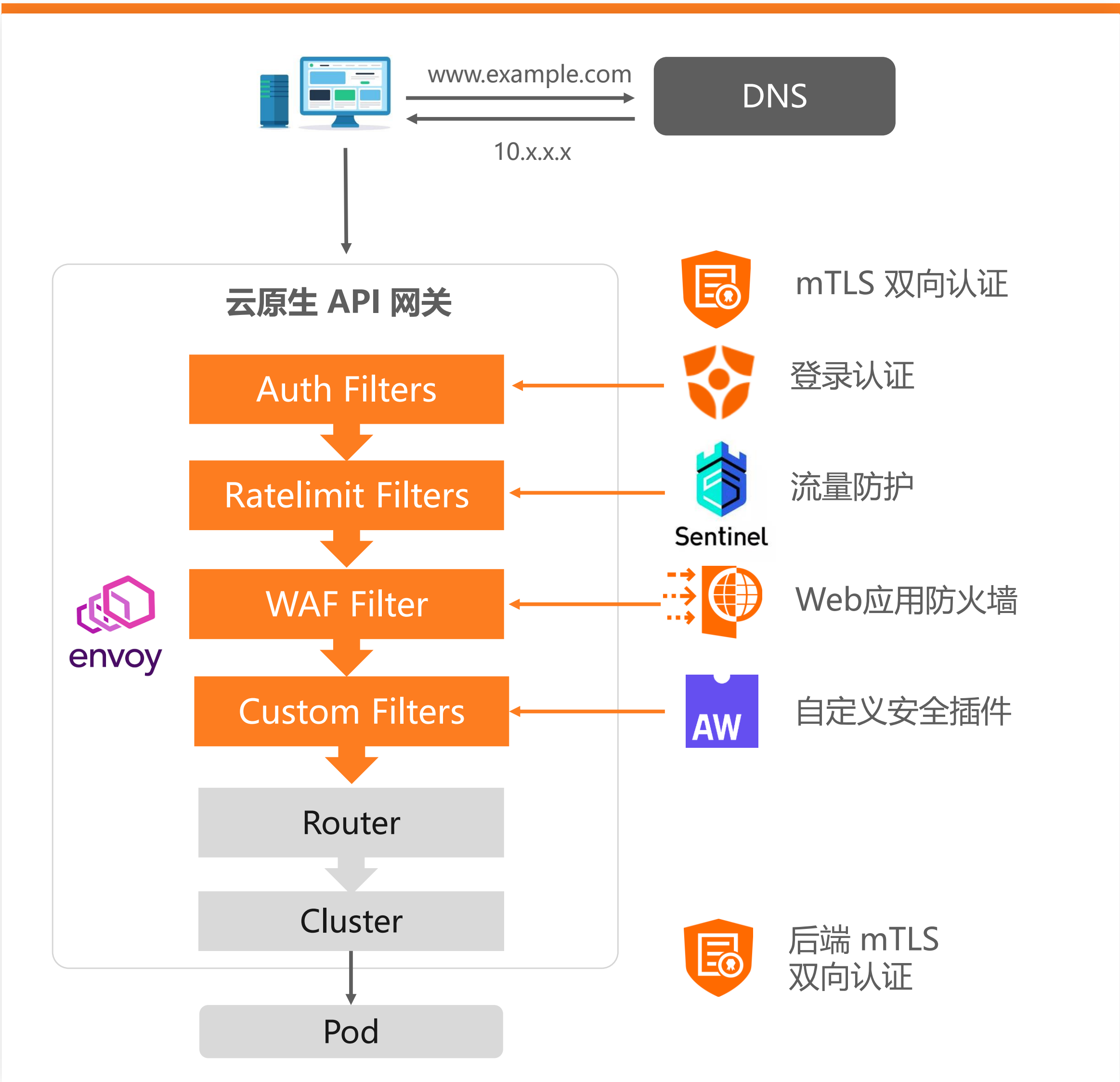
云原生API网关 – 高可用 (SLA: 99.99%)



- 网关自内部2020.5上线，已在支付宝、钉钉、淘宝、天猫、优酷、飞猪、口碑等阿里各业务系统中使用， 数年以来可用率100%，无任何故障。
- 历经多年双11海量请求的考验，大促日可轻松承载每秒承载数10万笔请求，日请求量达到百亿级别。

技术积淀已久，历经多年双11考验，每秒承载数10万笔请求

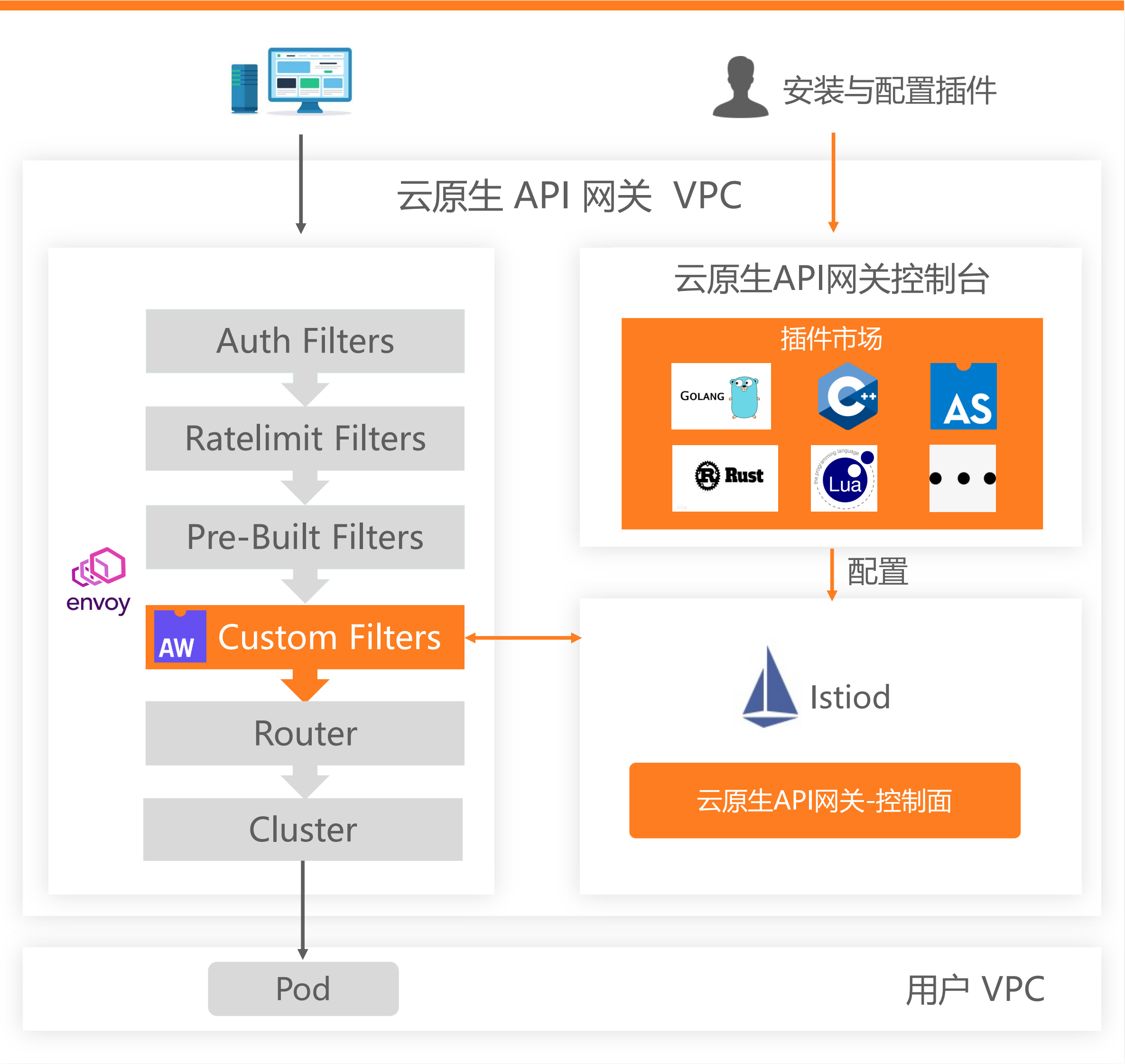
云原生API网关 - 安全能力



核心优势

- 消费者鉴权
 - 支持消费者认证&鉴权
- mTLS 双向认证
 - 集成阿里云证书服务自动更新
 - 支持 mTLS 双向认证，零信任
 - 支持硬件加速
- 登录认证
 - 支持JWT/OIDC/自定义多种认证登录机制
 - 集成 IDaaS 对接支付宝，淘宝等三放认证
 - 支持黑白名单
- 流量防护
 - 支持应用级和服务级流量控制
- Web应用防火墙 (WAF)
 - 更短用户的请求链路
 - 支持路由级防护能力
- 自定义插件
 - 提供默认安全增加组件
 - 支持多语言自定义扩展
- 内核优势
 - 采用数据面+控制面分离架构，防止控制面风险外溢到数据面
 - 采用WASM扩展机制，控制操作范围
 - 采用Envoy内核安全规则热更新

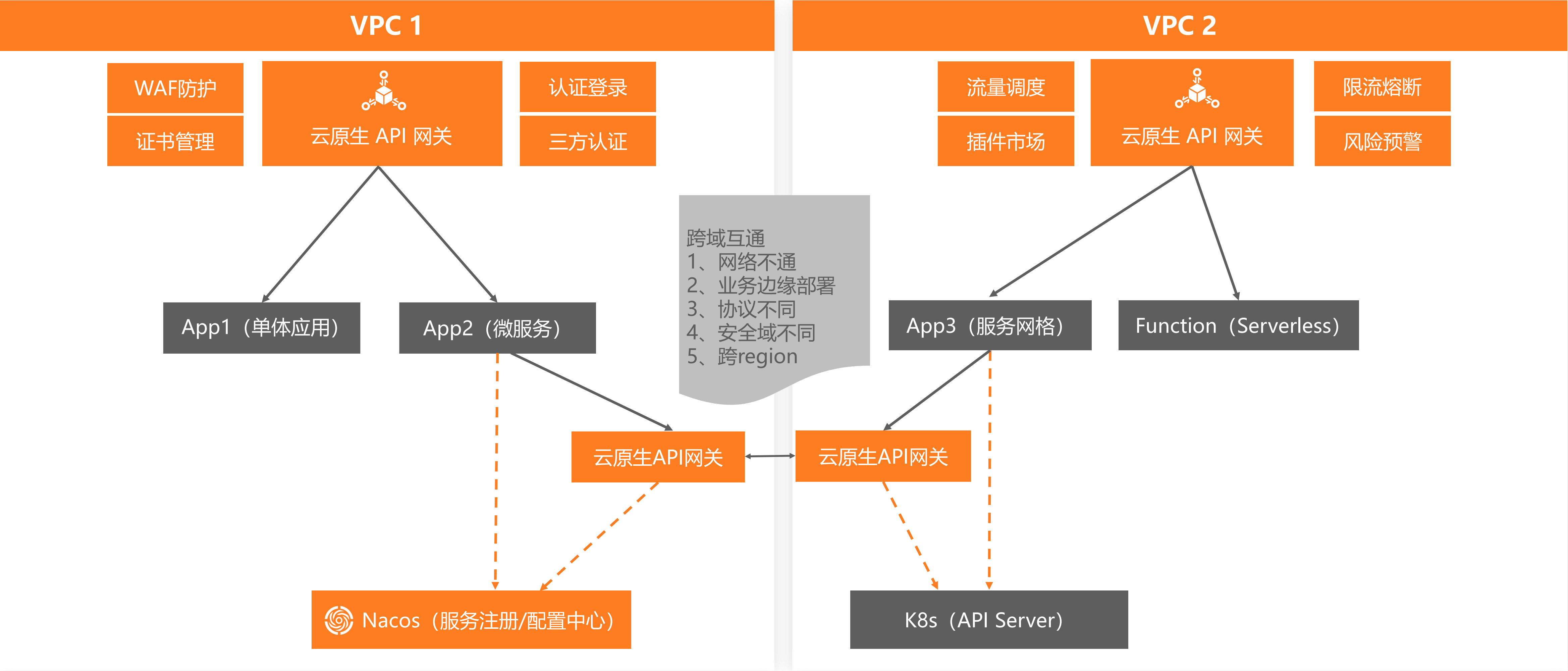
云原生API网关 – 插件机制（灵活扩展）



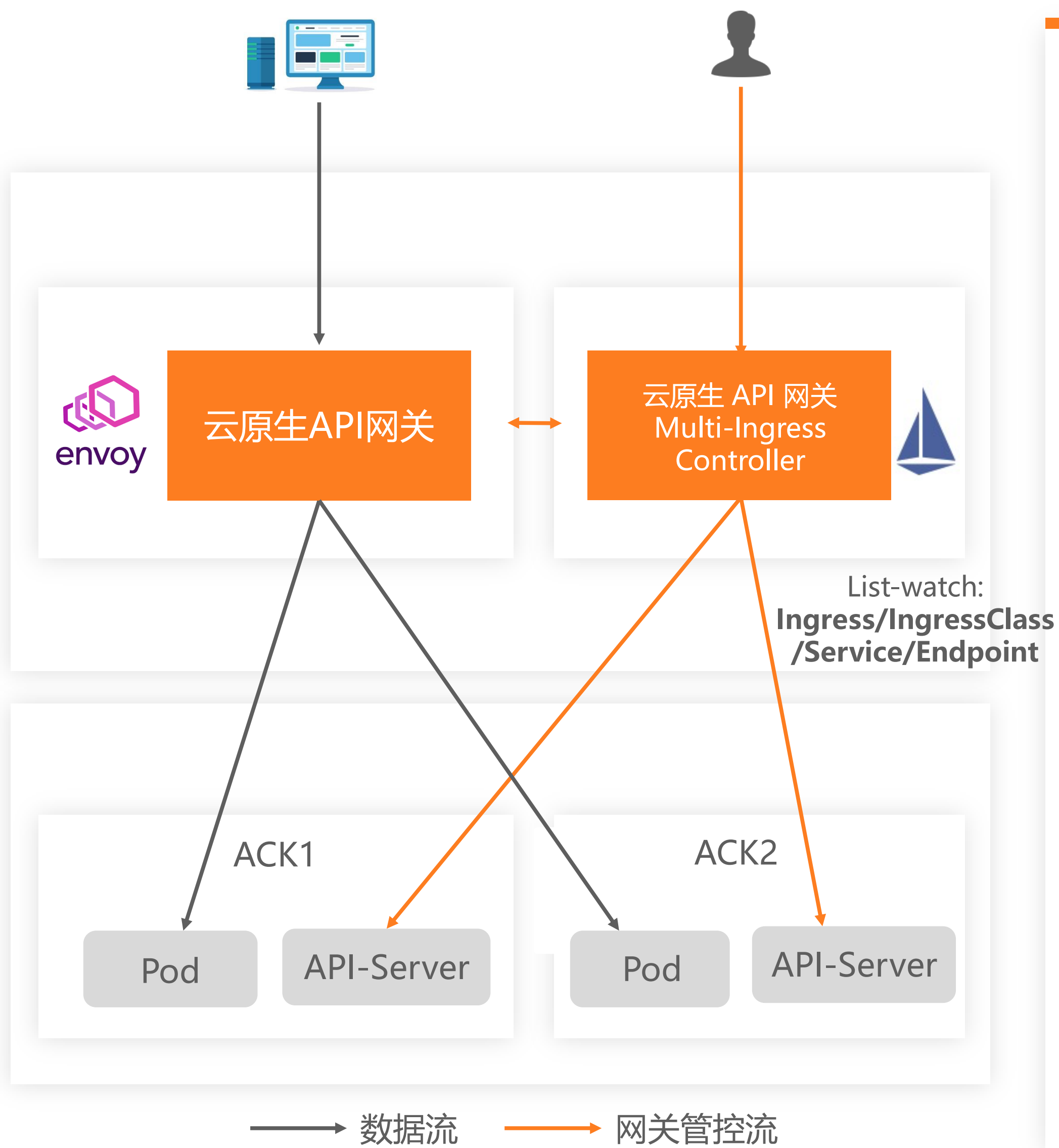
核心优势

- 借助WASM特性支持多语言扩展
- 提供在线 IDE, AIGC生成插件, 降低编写插件门槛
- 网关Wasm插件与开源Envoy 100%兼容, 不存在锁定
- 提供插件市场, 网关的二次扩展功能均通过插件提供给用户按需使用
- 插件采用热更新机制, 在沙盒中执行, 对网关自身稳定性无影响

附录 流量网关最佳实践

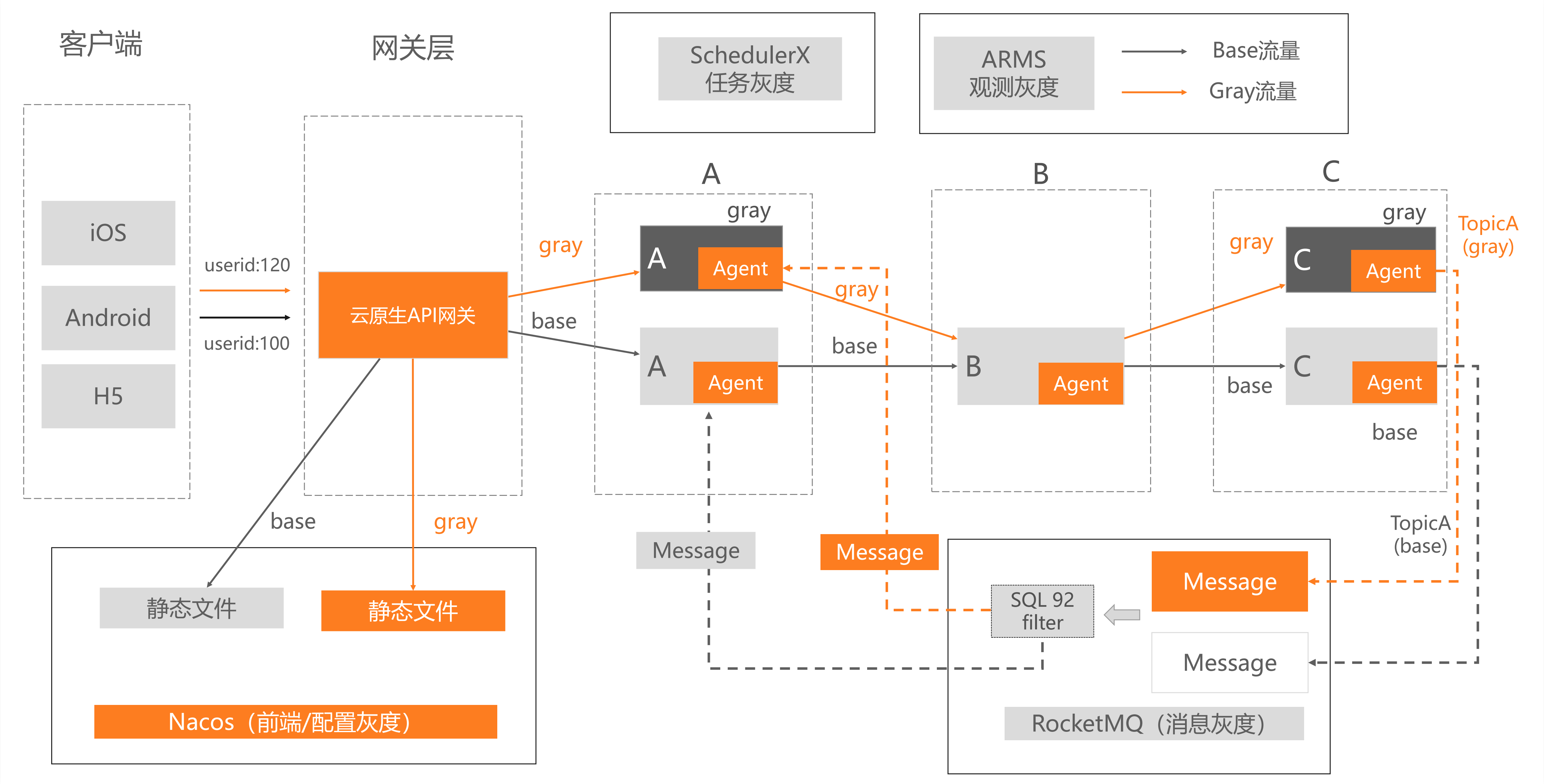


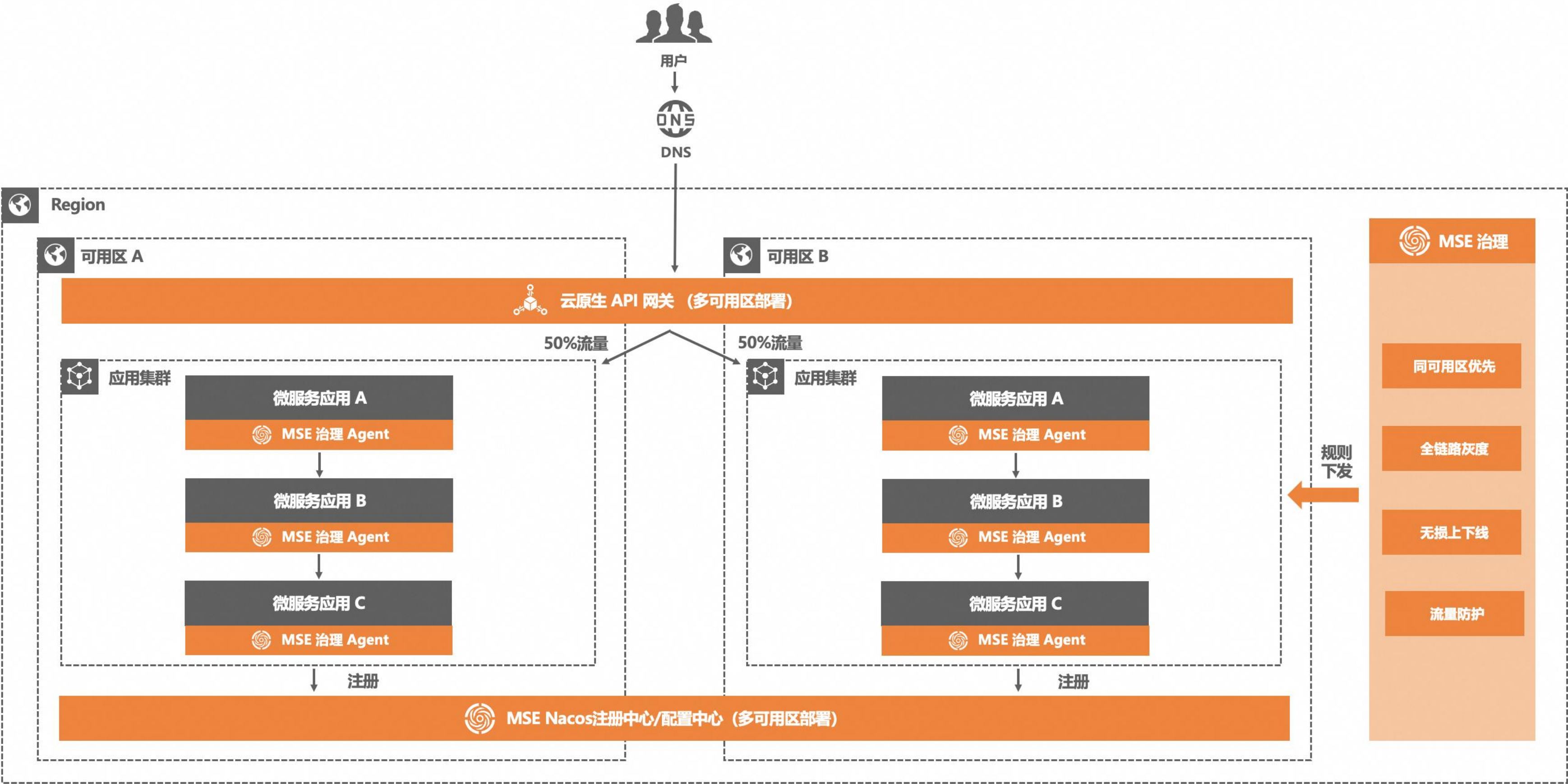
K8s Ingress



- 支持ACK/ACS集群内服务的自动同步
- 支持多ACK/ACS集群复用一個网关实例
- 支持 K8s Ingress / Gateway API 规范
- 支持 Nginx Ingress 核心注解扩展
- 支持 ACK One 多 K8s 集群容灾







方案优势

网域（CLB/NLB）层和网关服务层解耦，网域层具备逃逸机制

云原生API网关多可用区部署，对跨可用区的多个业务集群的请求实现高效负载均衡分配，单可用区集群故障时，可实现秒级故障转移。

一套注册中心，多可用区部署，可实现故障节点秒级自动剔除

接入微服务治理，可根据不同场景，在控制台上一键开启同可用区调用，支持设置节点数阈值，如可用区节点数超过50%时同可用区调用生效。